UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

UNIVERSITAT DE BARCELONA

Master in Artifical Intelligence

Master thesis:

Study of Deep Learning Models for Class-Agnostic Counting

Author:
David Hilazo Aguilera

Supervisor:
Javier Béjar Alonso

January 10, 2021

Abstract

In the field of object counting using computer vision techniques, multiple research projects developed highly performant models for counting specific classes. Nonetheless, none has achieved a deep learning model able to count any object independently of the class and very few have worked in that direction. We believe that achieving this class-agnostic counting model is possible by using deep learning methodologies such as Convolutional Neural Networks.

We have developed a set of Convolutional neural networks following different counting strategies to validate our hypothesis and achieve a class-agnostic counting model, as well as with the purpose of proposing the best strategy to achieve a model with these characteristics. The developed models, try several ways of approaching the counting task using common CNN architectures that work well in other environments and by analyzing the counting task from a human point of view while trying to mimic our image processing behavior. Furthermore, three datasets have been defined following different counting approaches for training and evaluating the developed models, even in unseen data to evaluate their generalization capabilities without requiring to be trained on it.

This master thesis proposes what we believe is the best approach for achieving class-agnostic counting as well as a new model improving the current state of the art in class-agnostic counting. We believe that the best approach is by developing a regression-based model with a spatial density output which splits the several tasks that take part in the counting problem into smaller and, if possible, supervised modules specialized in each sub-task. Moreover, this project offers a set of datasets that can be used in future works, fomenting this way the further research of a true class-agnostic counting model which doesn't require to be trained in unseen instances.

Contents

List of Figures

List of Tables

1 | Introduction

This master's thesis tries to solve the Computer Vision problem of **counting elements from an image**. With this purpose, several approaches were evaluated seeking for a Deep Learning model capable of counting elements independently of the target's class.

1.1 Motivation

With the start of the pandemic situation caused by COVID-19 in the year 2020, many projects for counting people in public areas have been promoted with the purpose of controlling their capacity and avoiding agglomerations of people to reduce the spread of the virus.

Counting elements from images has been studied in the field of Computer Vision since the early stages of this research area. However, as will be seen in Section 3, most of the developed projects for handling this problem have always been specialized in counting a subset of objects due to the complexity of the task. When working with real-world images usually several image characteristics can hinder the counting task, *e.g.*: too much noisy information from the background, irrelevant objects, low definition due to blurring, noise or objects obstructing the view. A good image preprocessing can help in solving most of these problems but what differentiates a relevant element from an irrelevant one? Obviously, it depends on the context we are working on, for this reason, defining a subset of elements to recognize has been a great approach until now. But this requires having a specific dataset for each task and subset; and training the model again with the new subset hoping that it will be able to recognize them. This is a tough process and, as the subsets get larger, the harder the training becomes. In fact, recent counting models usually work on counting 1 or, at most, 2 elements from an image.

Apart from the set of object classes to count, one must also consider the inner variance of the class (how different are objects within the same class) and if this variance has been taken into account when forming the dataset. Additionally, the model must be scale and rotation invariant to be able to recognize the objects in any environment.

As can be seen, the task of object counting groups the complexities of different computer vision problems into a single task. For this reason, the best approaches until now are limited to a subset of classes in order to sacrifice their class flexibility to gain a more reliable model in the different environments the class can be in. Some projects have been developed in the direction of achieving a class-invariant or class-agnostic model [14,15] however, no true class-agnostic model, which doesn't require explicit training on unseen classes, has been developed yet.

Automatic object counting has been used in the tasks of traffic and parking management, cell counting in medicine applications (*e.g.:* counting the amount of copies virus genome copies), animal migration control and, as mentioned previously, human crowd control through surveillance cameras, among many other applications. Developing a class-agnostic counting model will allow applying it in environments that interact with multiple classes such as autonomous cars (which requires counting pedestrians, cars, traffic signs, etc.), robotics or even warehouse product management; apart from being able to generalize all the previous applications into a single model.

In this thesis, we evaluate the different approaches for handling the counting task with the goal of discovering which is the best strategy to follow in a class-agnostic counting problem. Additionally, we try to define a model that improves the performance of the current state of the art in class-agnostic counting. We work under the hypothesis that a true class-agnostic model, that can count objects from any class by using a reference of the element to count, can be developed using Deep Learning methodologies. During the seek for this model, we will make reference to how we think humans are able to count any object and try to apply the same process with different Convolutional Neural Networks.

1.2 Contributions

The contributions of this thesis can be summarized in 3 main concepts.

The first contribution is the creation of three datasets for the counting objects task. The first one is a dataset created from the CIFAR10 which allows, thanks to its simplicity, a fast training of counting models using a regression-based direct counting approach. However, the dataset can be easily adapted to fit other approaches as well. The remaining two datasets are the ILSVRC2015 and the CARPK datasets transformed to be used in a spatial density counting task. All three datasets will be published to foment further investigation in the direction of class-agnostic counting.

Secondly, this project evaluates different strategies that can be followed for performing the task of class-agnostic counting. After comparing all strategies, a final approach is proposed as the best strategy to follow when developing a class-agnostic counting model, with some additional comments on how to improve its performance if more data is available.

Finally, the last contribution of this research is the proposal of a new deep learning model that improves the current state of the art in class-agnostic counting. As well as discarding multiple approaches that were evaluated and their performance was not better than the state of the art.

1.3 Overview

This thesis is split into 6 parts. Section 2 starts with a detailed explanation of the theoretical background required for the comprehension of the work done. Starting, in Section 2.1, with a description of what are Convolutional Neural Networks and how they work. Following up with a definition of pooling layers and their benefits, presented in Section 2.1.1. Next, Deep Residual Networks (Section 2.2.1) and Siamese networks (Section 2.2.2) will be presented as common CNN architectures that will be used as a reference when developing the models for this project. Moreover, another network architecture that is used as a component in the defined models is the Variational Autoencoder, which is explained in Section 2.3. For the last part of this section, an introduction to Transfer learning is given in Section 2.4 explaining how faster training can be achieved in deep learning models by using pre-trained weights.

After that, Section 3 presents an analysis of the current state of the art in the subject of automatic object counting and class-agnostic object counting. In this section will be discussed the different approaches followed since the early stages of the computer vision for counting elements research area, which specific techniques have been used, as well as what are the differences from this project. Afterwards, Section 3.1 will jump into the specifics of automatic object counting methodologies used by previous works.

Once the theoretical background is clear, Section 4 describes the relevance of having a dataset with high inter-class and intra-class variance for object counting, as well as presenting the three datasets, *Count CIFAR10* dataset in Section 4.1, *ILSVRC2015* dataset in Section 4.2 and *CARPK* dataset in Section 4.3, that will be used for training and evaluating the proposed models.

Section 5 describes in detail the procedure followed during the practical part of this project. More specifically, in Section 5.1 are presented the models defined by this thesis to overcome the automatic object counting problem following two different approaches. This section will include the characteristics and internal architecture of each of the counting models, as well as the reasoning behind the decisions taken to build each model. Later on, Section 5.2 will present the procedure that has been followed to obtained the final version of the previously mentioned models, in Section 5.2.1, as well as the experimental setup (in Section 5.2.2) describing the experiments that have been performed for comparing both approaches and obtaining a final model that is closer to achieve a true class-agnostic counting model. Moreover, the different metrics used to evaluate the models in the same conditions will be also explained in Section 5.2.3.

The results from the proposed experiments will be presented and analyzed in Section 6. Here, both approaches will be studied separately, starting by explaining the direct counting experiments

in Section 6.1 and then the spatial density counting experiments in Section 6.2. Finally, a comparison of both strategies will be provided in Section 6.3 along with the proposal of the model with best performance results.

To conclude, the final conclusions of this work will be presented in Section 7 describing what has been learnt from this project while comparing it with the previous work mentioned in the state of the art. Additionally, the possible future work will be mentioned describing the next steps that can be taken for achieving the desired class-agnostic counting model.

2 | Background

The presented thesis performs an evaluation of the problem of counting elements from images through different Convolutional Neural Networks (CNN) architectures. This section will give an overview of the theoretical concepts required for understanding the followed methodology, as well as understanding the contributions of this project.

First, in Section 2.1, the methodology behind the convolution filters will be described, as well as other common layers in CNN. Afterwards, Section 2.2 will mention a few common convolutional architectures, which are used as baseline for the defined models in this project. Additionally, Section 2.3 will explain the basic functioning of another artificial neural network called Variational Autoencoder which was used as a component in the developed models for obtaining an encoded version of the input images. Finally, Section 2.4 will describe what is Transfer learning and why it was so helpful to fasten the learning process of the trained models.

2.1 Convolutional Neural Network

A Convolutional Neural Network is a type of Artificial Neural Network which is commonly used for processing image input. Other Feed-Forward networks treat each input variable as being independent of the others without taking into account their spatial correlation. However, the spatial correlation between the pixels of an image is highly relevant in order to find patterns in the objects represented by the image.

Convolutional Neural Networks take into account the spatial correlation between input variables by defining what is known as convolution filters. The convolution filters are a set of weighted matrices that perform the convolution operation, shown in Equation 2.1, through each pixel in an image. Filters tend to be smaller than the input image allowing each one of them to specialize in a specific feature. In order to compute the convolution through every pixel in the image, it will start at an initial point, *e.g.* top-left, and slide down the image after each convolution operation until reaching the end of the picture.

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2.1}$$

Equation 2.1 shows a discrete representation of the convolution function where, given an input image I and a convolution filter K (also known as kernel) of size mxn, the convolution operation can be performed to obtain a feature map S. This convolution operation can be represented in a discrete way as the summation, for each axis in the kernel, of the product between the value of the image in position $(i-m, j-n)$ and the weight of the kernel in the coordinate (m,n). A visualization of the convolution operation in 2D matrices can be observed in Figure 1. The presented example shows the computed output for a kernel seeking for cross patterns in a binary image.

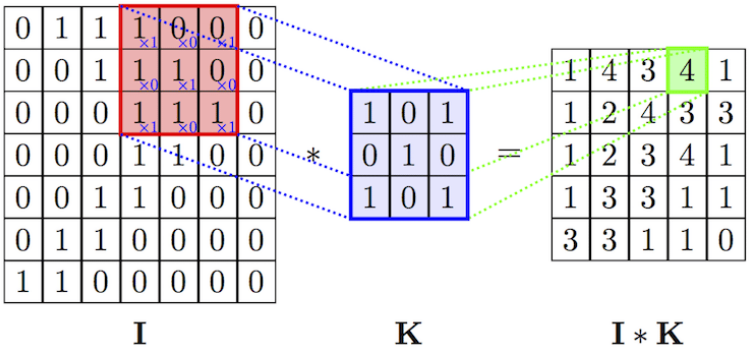


Figure 1: Convolution operation example in 2D matrices.

This operation, alongside backpropagation, will allow the learning of patterns in the image by changing the weights of the filters based on the supervised error. Once trained, each kernel will be a feature detector recognizing specific patterns in the image. However, in order to analyze the features of an image, the kernels should be as big as the feature to recognize. To overcome this problem, convolutional neural networks take the advantage of the Multi-Layer Perceptrons of stacking different convolution layers to recognize complex patterns by the aggregation of the simpler ones from the previous layer.

It must be said that convolutional filters can be used in many different environments, not only images, and can have multiple dimensions depending on their purpose. For instance, 3D convolution filters are commonly used on videos since they can compute the convolution operation in multiple frames at the same time. Nevertheless, in this thesis only 2D convolution filters will be used since the input to our model will be 2D images. Nonetheless, 2D images can have multiple channels depending on the colorfulness of the picture. The most common image color codification is RGB where each image is coded in three different channels (Red, Green and Blue). This number of channels will increase as we get deeper into the network since the output of a layer will have as many channels as kernels. Convolutional layers already take into account this fact and create an independent filter for each channel since some of the features may be present only in one of them.

The previously mentioned convolution function, shown in Equation 2.1, can be slightly modified with the different configuration parameters of the convolutional layers. These parameters include the *padding size* applied to the original image, which helps finding patterns in the borders of the image, and *stride*, which describes how much the filter will be slid through the image after each convolution operation, among many others. The common values for padding and stride are 0 padding and stride 1. Higher stride values will allow reducing the dimensionality of the input image since it skips the computation of the convolution function in the pixels in between.

The dimensionality reduction is useful since it fastens the computation and allows reducing the complexity of the problem until reaching the final layer where an answer must be given (E.g. a class number indicating if the picture contains a cat or a dog). However, convolutional layers with high stride values are not the only way of reducing the dimensionality of our input. A common alternative is the usage of Pooling layers.

2.1.1 Pooling layers

As briefly mentioned previously, Convolutional Neural Networks often use pooling layer for reducing the dimensionality of the input to speed the computation as well as providing a more robust representation by focusing on the relevant features of the input. Similarly to convolutional filters, pooling layers also have defined a filter size as well as a stride to define their sliding through the input. Pooling layers work by sliding a filter through the input image and aggregating its values into a single. The procedure followed to decide which value should be kept depends on the type of pooling layer, of which we can distinguish:

- **Max pooling**: From all the input features contained inside the filter in the current iteration it will keep the one with the highest value discarding the others.
- **Average pooling**: Performs an average of all the input values contained inside the filter.
- **Adaptive Max/Average pooling**: Adaptive pooling works as any other standard pooling but instead of defining the kernel size and stride manually, it will define them based on the input and the desired output shapes. This kind of pooling layer have been used for defining neural networks architectures where the input size is unknown and can vary between instances. This allows a more flexible architecture, however, depending on the size difference the performance could drop considerably.

The pooling layers listed above are the "standard", many other custom pooling layers have been made over time (such as Min pooling), however, the most efficient and therefore the most commonly used, are the ones mentioned previously.

One advantage of pooling layers is that they don't have any parameter to learn from gradient descent. Thus, backpropagation will skip these layers passing the gradient to the previous layer. Nevertheless, for max-pooling layers, the gradient passed to the previous layer will only change the feature with the highest value. While average pooling layers will equally divide the gradient through all features as expected.

2.2 Common CNN architectures

In this section will be described the most common Convolutional Neural Networks that have been used as baseline for the definition of the models in this thesis. The characteristics for each of the following architectures will be explained as well as the structure, benefits, disadvantages and their current main applications in real-world tasks.

2.2.1 Deep Residual Networks

Deep Residual Networks, or ResNets, have been developed in 2015 by He, K., *et al.* in their paper called *Deep Residual Learning for Image Recognition* [12]. In this work, they studied if the deeper the network is, the better are the obtained results. After training different sized networks on different datasets, they discovered that the depth of the network wasn't directly related to the model's performance and it depended more on the task to be performed. Based on this fact, they defined the core component of Deep Residual Networks the "residual learning block", shown in Figure 2, which groups a stack of layers and fastens the training stage thanks to defining a skip connection.

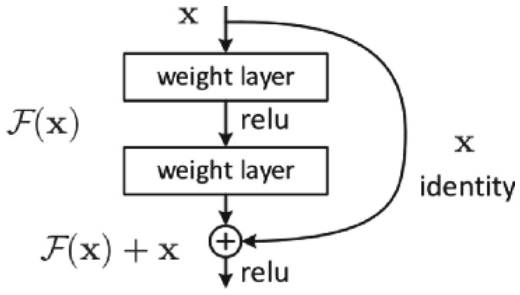


Figure 2: Building block scheme for performing residual learning in neural networks, obtained from the *Deep Residual Learning for Image Recognition* [12] paper.

Instead of hoping that a stack of layers will provide a better feature mapping, this building block allows skipping blocks of layers thanks to the identity connections. This way, in case that the optimal mapping has been already achieved by the previous block it can be kept as the optimal mapping by setting the non-linear layers inside the residual learning block to zero. Although the same identity

weights could indeed be fitted directly into these layers, it has been proved that learning the identity mapping is harder than pushing the weights to zero. Therefore, this connection provides a faster learning for deeper neural networks. Additionally, thanks to the residual connection, it lets the network to decide which is the best architecture for each task.

One question that might come to mind is that since it can skip convolutional layers, the performed convolutions will probably change the dimensionality of the input producing a miss-match with the values from the residual connection. In order to fix this problem, two approaches can be followed. The first one is to add zero padding to the outsides of the convolved input $F(x)$ to all the different dimensions to make it fit the same shape as the input value. On the other hand, the second option and the most common one is using 1x1 convolution filters to expand the dimensions and up-sample the data.

Based on the idea of the residual learning building block, many different architectures can be built by stacking these blocks. The most well-known ResNet architectures are ResNet-34, ResNet-50, ResNet-101 and ResNet-152, where the number indicates the number of layers it has. Figure 3 shows the layer distribution of a ResNet-34. As can be observed, apart from the usual convolutional layers, the network also presents dimensionality reduction either by applying pooling layers or by applying a stride of 2. After all the residual blocks, a set of fully-connected layers can be added to fit the desired output for the current task.

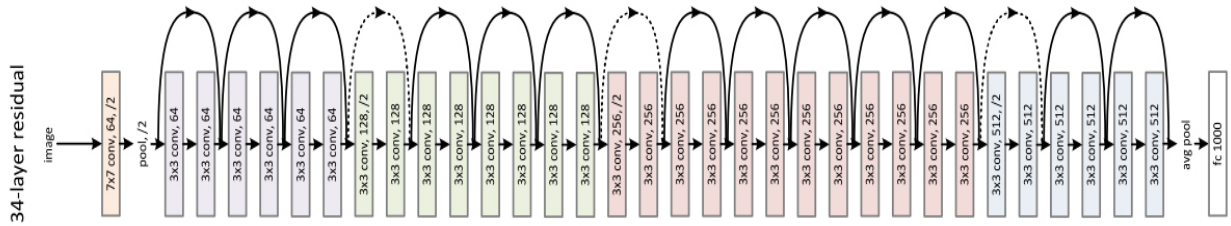


Figure 3: ResNet-34 layer architecture. The dotted shortcuts represent an increase of the input dimensions.

As mentioned previously, this type of network architecture provides a faster learning for deeper networks and, thanks to the residual connections, the network can adapt to a wide range of tasks. Nonetheless, residual networks also have their disadvantages. The first drawback is that since it is a deep network with many layers, the network has a lot of trainable parameters that have to be stored in memory. For example, the ResNet-34 model has more than 21 million parameters which are considerably less than other common models like VGG-16 but still are quite a great amount of parameters that requires having a powerful machine to train it. Additionally, although ResNet models achieve a good performance in most tasks, they are usually surpassed by other models with a specific structure built based on the task. For this reason, ResNets are usually used as backbones for building more specific models by truncating the network or modifying some of its layers.

2.2.2 Siamese networks

The problem of comparing different input data has been approached from different angles by the computer vision research community. The most common way of doing that is by using a pattern recognition neural network and returning the final class of the given instance. However, in 1993, Bromley et al. published a paper called *Signature verification using a "Siamese" time-delay neural network* [10] where they defined the first Siamese neural networks in charge of validating if two given signatures were the same or not.

Siamese neural networks (SNN) are a type of network architecture that is composed of two or more identical networks where each one has a different input value (This is the theoretical behavior, in practice the same network is used while passing it different inputs). This type of architecture can

be applied to both fully connected and convolutional neural networks and, although the behavior is the same, this section will focus on the convolutional siamese neural networks.

This kind of networks work by comparing the feature vectors of each input and compute their similarity. Thanks to that, siamese neural networks can be easily adapted to new input data since it returns a similarity metric instead of the prediction of a class, where an increase in the number of classes would imply a change in the network's architecture. To achieve this similarity metric, siamese networks can use different loss functions based on the distance of the feature vectors:

- **Triplet loss:** works by defining an anchor of what is being tried to be recognized attracting instances from the same class (positive instances) and repelling those instances from other classes (negative instances).
- **Contrastive loss:** works similarly to Triplet loss but without the definition of anchors and doing pairwise comparisons instead.

This architecture allows that Siamese neural networks can be used for classification, verification, object tracking tasks, among many others.

The analysis of objects through different angles improves their recognition thanks to being able to analyze more information about the same object. These different angles of an object could be stacked together and analyzed as if it was a video sequence however, the researchers from the Xidian University with help from Universities in Pakistan opted for a different approach in their work about *A Deep Siamese Convolution Neural Network for Multi-Class Classification of Alzheimer Disease* [16] where they used a Siamese convolutional network for classifying dementia stages.

Using this type of network on verification tasks allows the network to perform a pair-wise (or higher) comparison of different instances to ensure if they are related to the same class. An example of this kind of application could be the one mentioned previously from Bromley *et al.* or the *SigNet*, its later improved model for signature verification presented in the work from Sounak *et al.* named *SigNet: Convolutional Siamese Network for Writer Independent Offline Signature Verification* [18].

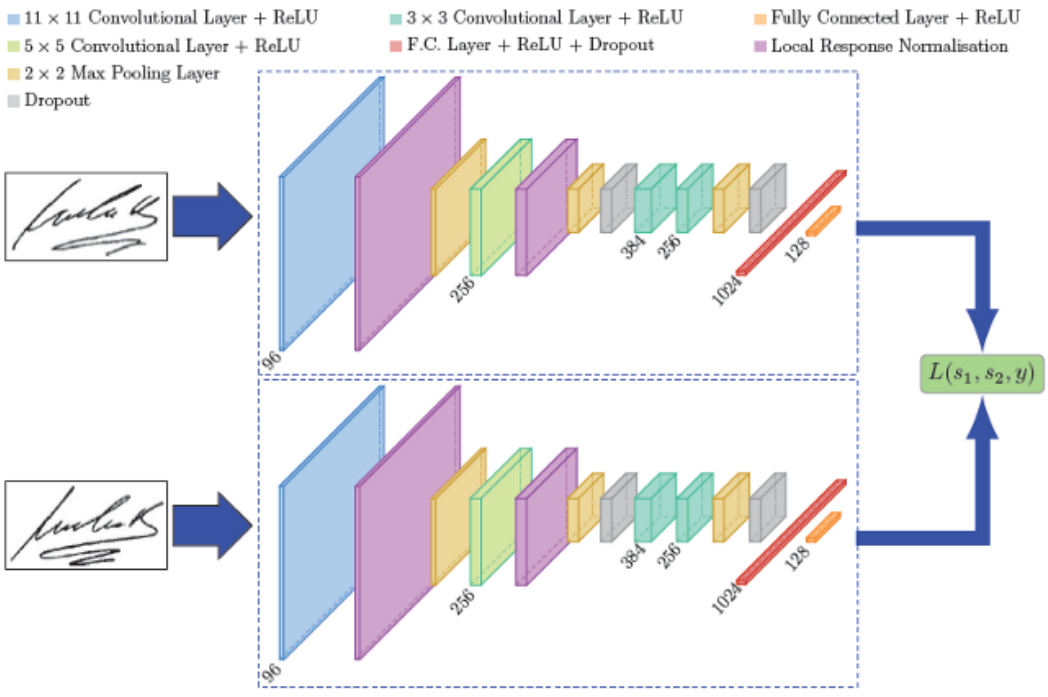


Figure 4: *SigNet* [18] model architecture for signature verification.

Finally, the application example that is more related to this project is object tracking where an exemplar image is given and it is compared against a whole frame of a video to locate its coordinates. An example of this use case is the work from Bertinetto, *et al.* on *Fully-Convolutional Siamese Networks for Object Tracking* [17]. As will be explained in later sections, this kind of architecture where a template image is given to be located in a separate image, will be the same behavior that will be used on the search of a class-agnostic object counting model.

2.3 Variational Autoencoder

Another type of artificial neural network, which is used as a component to build the defined models for this project, is the Variational Autoencoder (VAE). This section will provide an overview of what they are and how they work without getting into too much detail.

Before explaining what is a Variational Autoencoder, we must understand what is an autoencoder. An autoencoder is a neural network whose purpose is learning a shorter representation of the input data, also known as latent variables or latent representation. In order to do that, autoencoders make use of two sub-networks, the encoder and the decoder, as can be seen in Figure 5. As with Siamese neural networks, the autoencoders can be used with different types of data, however, this section will be focused on its usage on images.

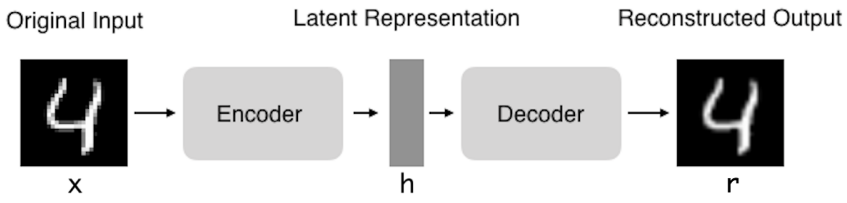


Figure 5: General structure of an autoencoder.

The encoder is a neural network that reduces the dimensionality of the input space until reaching the desired size for the latent representation. This compressed version of the input must be representative enough to describe the input for the decoder. Each latent vector represents a point in what is known as latent space. A latent space is a different representation of the data where instances from the same class are closer, thanks to their shared attributes, than instances from different classes. Therefore, it is able to capture the structure of your data.

On the other hand, the task of the decoder is reconstructing the input data only by using the latent variables, *i.e.* through the encoded values. Because of that, both networks are trained by computing the loss between the given input and the reconstructed output which makes these networks a powerful tool since they don't need supervised data because the input is already the ground truth as well.

The main drawback from autoencoders is that they are prone to overfitting since it doesn't take into account the organization of the latent space, trying to achieve the least loss possible independently of the encoded representation. For this reason, Variational Autoencoders have been defined with the purpose of solving this problem. A VAE works like an autoencoder but with a regularized training stage for avoiding overfitting and obtaining a better latent representation. This is achieved by adding a new step between the encoding and the decoding called the reparametrization trick.

The reparametrization trick is based in, instead of encoding the input as a point in the latent space, it is encoded as a distribution in the latent space by returning its mean μ and covariance σ. Then, the reparametrization step helps the regularization by interpreting this distribution as a standard normal distribution and sampling a point from it. This normalization prevents the

model from encoding overfitted points and achieves a more meaningful representation of the latent variables.

The encoding of an input is not supervised, which means that it is difficult to provide semantic meaning to each of the encoded features. For this reason, vanilla Variational Autoencoders are not used for generation purposes since it is hard to generate a specific object through its manual description in a latent space. Variants of this model, such as Conditional Variational Autoencoders, provide more control over the latent representation which helps the decoder of the model in being used as a generator model. Thus, Variational Autoencoders are usually used when a smaller representation of the input is wanted which describes the different attributes of the image. Nevertheless, we will not get into detail about Conditional Variational Autoencoders since this project only uses the VAE as an encoder for a given image.

2.4 Transfer learning

Many resources are required for training a deep learning model. In order to fasten the learning process, a common practice is using what is known as transfer learning.

Humans usually are able to learn new tasks by using the learnt knowledge from previous tasks. Transfer learning uses the same idea by training a network on a general task, such as object classification, and then using its pre-trained weights as an initialization for the network of the task to be done, as could be object counting. This avoids wasting resources on training the model from scratch each time to learn the basic feature mapping and achieves a faster convergence of our model. For example, a Convolutional Neural Network can use transfer learning by using the pre-trained weights for object recognition and avoid learning how to recognize lines, curves, etc. Instead, it can focus its resources on computing the gradient for more specific feature mappings required for its task.

The more general is the original task, more general will be the learnt features, therefore, more problems will be able to use its weights as transfer learning. On the other hand, if the original task is more specific, fewer models will be able to use transfer learning from it but it will fasten the learning from those who can.

In order to use transfer learning the model architecture from both the original task and the current one must be the same. For this reason, it is a good practice to use common architectures, importing its pre-trained weights and then modifying its architecture to fit the current task, for example by cutting the parent network and using only a group of layers, adding new layers at the end of the network, etc.

When using transfer learning on convolutional neural networks, it is important to know how are distributed the recognized features throughout the network. The convolution filters in the first layers of the network provide a general analysis of the given input like, as previously mentioned, the detection of lines, curves, corners, etc. Nonetheless, as we get further into the inner layers of the network, the features start being more specific and task-oriented. Most networks will use similar features for the first layers of the network due to their generalization, because of that, it is a common practice to freeze their weights to prevent computing the gradient on them which will fasten the training stage since fewer parameters will have to be updated and bigger will be the change on the remaining ones.

3 State of the art

Computer vision is a research area that uses artificial intelligence to train algorithms in order to be able to interpret the visual world. Until the appearance of computer vision techniques, machines could observe the world through the usage of digital cameras but were not able to understand it.

The first steps into the computer vision field were applications where the used features to recognize the elements from an image were what is known as "hard-coded", in other words, they were specifically defined for the current task without being able to generalize to any change in the brightness, rotation, scale, etc. Little by little, the learnt features achieved to be brightness or rotation invariant but still not general enough to work on different classes. With the creation of artificial neural networks and, more specifically, convolutional neural networks, computer vision algorithms started to learn features able to generalize on several classes and image characteristics, even in unseen data. This propelled the computer vision field into a wide variety of applications which previously were too difficult to work on and get proper results, such as image classification, object recognition, image segmentation and, among all of them, object counting.

When a dataset used in any of these applications contains elements from only one class in each sample and the classes are mutually exclusive, convolutional models can easily discriminate multiple classes. But, when different classes of objects are shown in the same image with different angles and scales (as happens in the real world), the task of recognizing the elements of an image becomes more complicated since it must first locate the different objects and then recognize them to discard the irrelevant ones. In fact, the task of object counting not only has to deal with this problem when working on real-world images but also has to keep track of how many elements were recognized.

The problem of counting objects using computer vision techniques has been approached from different angles over the years. The first approaches have been done by using more basic computer vision methodologies like Haar-like features, Bag of visual words or Histogram of Oriented Gradients (HOG) for extracting features from the images and then using regression models (such as Support Vector Machines or Random forests) for counting the positive matchings. These approaches had many disadvantages like being scale and orientation-dependent, being able to only count specific elements due to the bias of the defined filters, having false positives, or finding difficulties to count in crowded images.

As deep learning models gained popularity, many projects have been developed proving that there are multiple ways of completing the task. The different strategies that have been followed over the years can be split into two groups based on their methodology. The first group are the techniques which count by finding object instances in the images through visual detectors and provide as output the bounding boxes with the detected objects. These methods are known as detection-based counting. The alternative are the regression-based counting methods, which are known for learning a mapping between the image features to a scalar value representing the final count. This scalar value can be predicted directly by the network, which will be defined as direct counting, or represented through a spatial density map with a scalar value on each cell of the map, which in this project will be referred to as spatial density counting.

As mentioned in the introduction in Section 1, there are multiple problems that make object counting a hard task. For this reason, most developed projects either count a small set of specific objects or count too broad groups [13] to improve their performance, some cases even count elements in general without specifying any particular object [14]. The main objects of interest for specific object counting have been: people (for crowd counting methods) [1–4, 11, 15, 20–23, 25], cells (for counting blood cells or virus genome copies) [4–6, 15], cars [4, 7, 11, 15, 19, 24] and animals [4, 8], among many others.

Among all these objects of interest, crowd counting is the one that attracted the most attention from researchers due to its direct application in urban places and activities, such as in sporting events, capacity control in public areas, political meetings support (either rallies or protests), etc. Thanks to having diverse clear applications that can be visualized by investors without a technical background, it also attracted many sponsors who fomented its investigation.

In order to perform crowd counting, we must take into account the characteristics of the human body. The human body has a complex shape which can vary depending on the posture of the person. Because of that, different detection-based approaches have been followed over the years where the human body is divided in substructures to decrease the complexity of the task. These approaches can be divided in the following groups:

- **Monolithic detection:** Finds people instances by training a classifier that tries to locate the full-body of the person. This approach has been used both with and without deep learning techniques with the usage of the previously mentioned feature detectors: Haar wavelets, Histogram of Gradients, etc. Following this strategy has several drawbacks, among which the most obvious is a deterioration in the system performance when the given image presents occlusion of part of the body.

- **Part-based detection:** Instead of looking for the whole body, part-based detection focuses on a section or multiple sections of the human body such as the head and shoulders or legs. It is highly recommended to use multiple sections of the body since it avoids miss-matching them with other common objects.

- **Shape matching:** In painting, it is a common practice to define the proportions of the human body with a set of circles and ellipsoids. The same idea has been followed in Shape matching where the human body is detected through a combination of ellipsoids and then counting the matched instances.

The three detection-based crowd counting approaches can be visualized in Figure 6. From left to the right, the presented examples correspond to monolithic detection, part-based detection and shape matching.

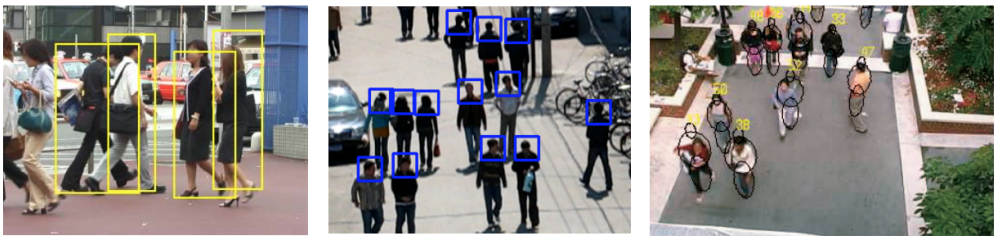


Figure 6: Examples of detection-based approaches in crowd counting.

On the other hand, regression-based counting alternatives for crowd counting extract the features from crowded images and then maps them into either a scalar value or a density map to show the areas with higher density, as shown in Figure 7. When comparing this approach with the previous ones, it presents a better performance on images with dense crowds or when there is background clutter.

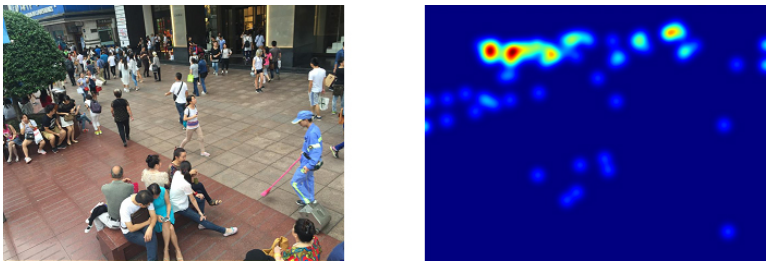


Figure 7: Example of a regression-based approach in crowd counting with density map as output.

Nevertheless, all previously mentioned models have a clear drawback which is their lack of flexibility and precision for counting a variety of specific elements. To overcome this disadvantage, more recent methods have been working on either increasing the number of classes a model can count, add adaptive modules to the existing networks to improve their performance in unseen environments, or creating class-agnostic counting models, like the goal of this thesis.

A successful project in the direction of building a class-agnostic model is the one developed by Lu, *et al.* named *Class-Agnostic Counting* [15] with funding from DeepMind Scholarship. It achieves to develop a deep learning model called Generic Matching Network which, from the image to be evaluated and a template of the object to be counted, it outputs an image with a Gaussian where each element has been located, therefore it is a regression-based counting model. After being trained with the ILSVRC2015 dataset, it works as a good initialization in unseen datasets. Although it obtains promising results in a wide variety of class objects and a fast adaptation on new classes, it only works as initialization and still requires to be trained on unseen data. This thesis differs from this project by trying to achieve a true class-agnostic model that doesn't require to be trained on unseen data thanks to obtaining a precise description of the given template. Moreover, a comparison of previously mentioned strategies for automatic regression-based object counting will be performed. To do that different models have been tested in order to propose which is the best approach.

3.1 Object counting methodologies

Once given the overview of the different approaches that have been followed for counting elements in an image. This section will get into detail about which deep learning methodologies were used by recent previous works, as well as the reason why this thesis will focus on comparing only regression-based counting approaches.

As briefly mentioned, there are two main ways of developing deep learning models for counting, based on the desired output. The first option is using detection-based counting where the output is represented by a set of bounding boxes indicating the coordinates of the found element. In order to achieve these bounding boxes, the most common approach is using Region Proposal Networks (RPN) as in [19, 20].

Region Proposal Networks are a type of neural network which is embedded in object detection models with the purpose of proposing a set of bounding boxes where the relevant objects might be located. As illustrated in Figure 8, showing the architecture of *Faster R-CNN* which has an embedded RPN module, Region Proposal Networks work from a preprocessed feature map that has been computed by a backbone network on the image to be analyzed. Based on this feature map it computes two different outputs: four regressed scalar values which define the coordinates of the bounding box (known as anchors), and a classification value describing whether if an object has been found in that region or not. This output will generate multiple region proposals where not all of them will be valid (most will probably overlap with each other), because of that, the output will be passed into another module to compute the final region proposals. In the example presented in Figure 8, this task is performed by a *Fast R-CNN* which, from the feature map and all the region proposals, selects the final regions.

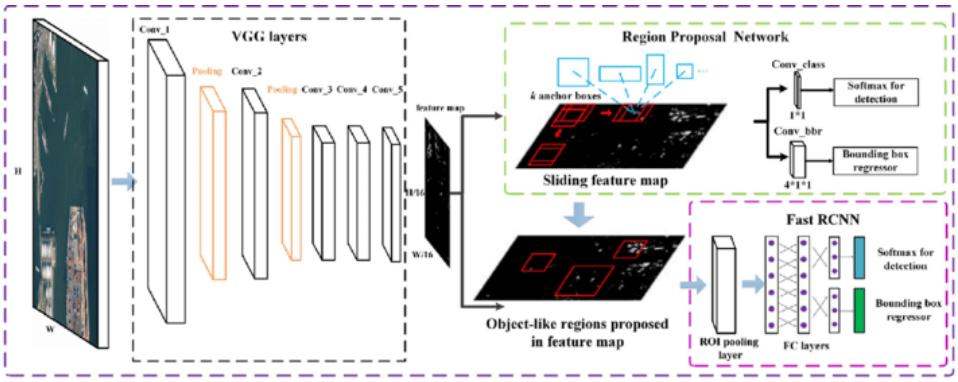


Figure 8: *Faster R-CNN* module architecture composed by a cut *VGG* network, a RPN and a *Fast R-CNN*.

Learning the anchors and the classification values is achieved thanks to the backpropagation from the ground truth annotations. It must be said that region proposals can also be achieved manually by using sliding windows and a set of handmade anchors. Each defined anchor will be computed at the location of the sliding window and the classification value is obtained through the computation of the *Intersection-over-Union (IoU)* between the area inside the anchor (A) and the area inside the ground truth annotated region (Gt), following the Equation 3.1. After that, a threshold value (*i.e.: 0.7* is applied in order to discard the invalid region proposals.

$$IoU = \frac{A \cap Gt}{A \cup Gt} \tag{3.1}$$

Region Proposal Networks have the advantage of avoiding the computation of all possible regions for the handmade anchors. Nonetheless, this technique also has a drawback which is having to define a maximum amount of region proposals in an image, limiting the number of objects that can be found in an image. Moreover, different studies [21] showed a decrease in the performance of the detection-based counting models when the evaluated image had a higher density of objects, due to the difficulty of finding region proposals in objects that are too close together and the possibility of creating regions containing multiple objects.

The alternative procedure is using regression-based counting, where the expected output is a scalar value with the predicted count for the image. As mentioned in the previous section, this output can be presented as a single scalar value or as a 2D spatial density map. The most common procedure is using as output the spatial density map, which can be obtained by using convolutional neural networks that encode the input image while down-sampling it to aggregate the different features of the input. From this point, there are two valid options, either up-sampling the encoded features to obtain the final heat map or using 1x1 convolutions to reduce the dimensionality while keeping the scale of the image. Many network architectures fit these requirements, in the case of up-sampling the feature extraction, U-Nets are the most common option for this task as in [11].

Obviously, this is the main procedure and can be varied in multiple ways. One common option is using multi-column [22, 23] or multi-resolution [24] convolutional neural networks to obtain a scale-invariant model through the analysis of high-level and low-level features.

Multi-column convolutional neural networks are a type of CNN which architecture is formed by two or more convolutional branches with different kernel sizes. This kind of architecture helps the network in the process of feature extraction when the elements present different sizes or proportions due to the camera lens. Figure 9 presents the architecture of a Multi-column CNN (MCNN) for density map estimation from the *Single-Image Crowd Counting via Multi-Column Convolutional Neural Network* [22] paper.

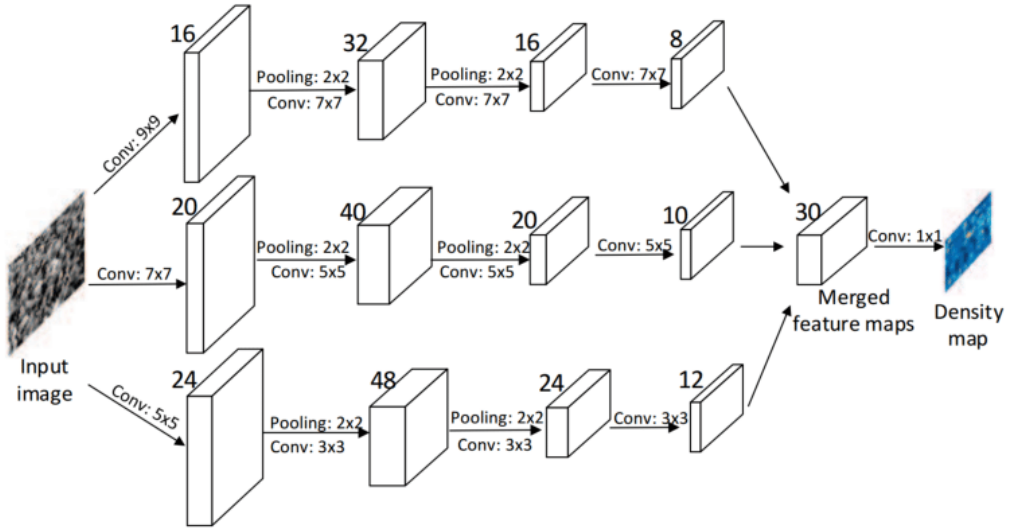


Figure 9: *Multi-column CNN* architecture from *Single-Image Crowd Counting via Multi-Column Convolutional Neural Network* [22].

Alternatively, Multi-resolution convolutional neural networks analyze the image by rescaling it and feeding all versions of the picture into separate branches of the network with the same filter size. This approach is more similar to the previously explained Siamese Networks but with the difference that the input image is the same and the weights from each branch is different. An example of a Multi-resolution CNN is shown in Figure 10 where the architecture of the *Hydra CNN* from the work on *Towards perspective-free object counting with deep learning* [24].

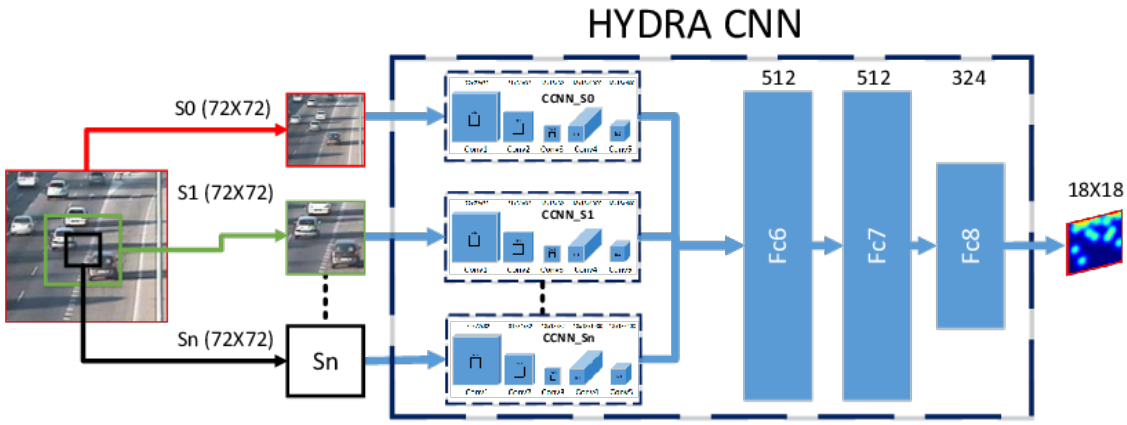


Figure 10: *Hydra CNN* architecture from *Towards perspective-free object counting with deep learning* [24].

In order to obtain the count from a spatial density counting model, one can either sum up the predicted counts on each pixel or count the local maximums in the density map depending on the structure of the output. It must be said that there are as many network architectures and approaches as projects, but this section describes only the most popular.

On the other hand, the network architecture for direct counting with a regression-based strategy is the usual convolutional neural network where through the usage of convolutional layers, pooling layers, and linear layers at the end, the input gets reduced until reaching a single value. These models are harder to train since it is the network's job to find the correlation and mapping between the input image and the desired count. One example of a regression-based model with direct counting is the one presented in *Deep People Counting in Extremely Dense Crowds* [25] from Chuan, W. *et al.*

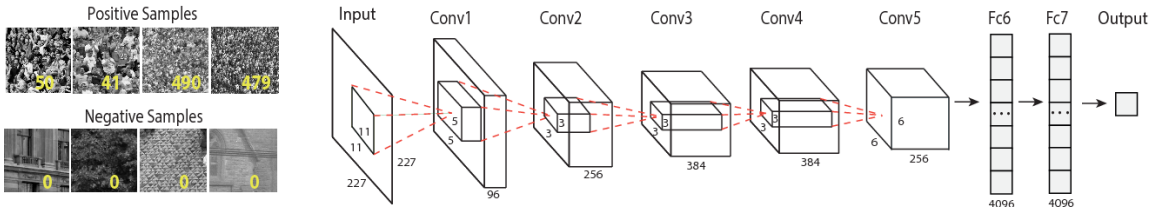


Figure 11: Deep model architecture for direct counting from *Deep People Counting in Extremely Dense Crowds* [25].

Both detection and regression-based approaches are not mutually exclusive. An example would be the work from Stahl, T.; *et al.* on *Divide and Count: Generic Object Counting by Image Divisions* [14] which follows a hybrid approach where the image is split into regions and then a count is regressed from each region.

This project evaluates the performance of models under both regression-based approaches with the purpose of analyzing their advantages and disadvantages, as will be explained in the next sections. The designed models are wanted to be as general as possible working in several classes and all kind of input images, independently of their spatial density, brightness, scale, etc. Detection-based counting approaches have been discarded for the analysis since they provide less flexibility to different inputs than regression-based models and, in case the location of the objects is wanted, the bounding boxes of the elements can be computed from the width of each local dense region from the regressed output.

4 | Data

The definition of a dataset defines how the problem will be faced and the restrictions that the network's architecture must follow to work with it. This thesis evaluates the performance for both regression-based approaches (direct counting and spatial density counting) through the creation of a dataset for training each approach. In this section will be described the datasets used for training and evaluating the defined models to complete the task of counting objects under different scenarios, regardless of the class. In order to correctly evaluate the performance from each model, every dataset is split into separate train, validation and test data splits to be able to detect when the model is being overfitted or underfitted and provide a final unbiased performance result. Moreover, a third dataset (CARPK) has been added to evaluate the performance of the spatial density counting approach in unseen data.

When defining a dataset for deep learning, it is important to keep a high intra-class and inter-class variance in order to let the model learn the widest range of features for each class. Other machine learning algorithms prefer a low intra-class variance and high inter-class variance in order to discriminate easier the different classes. However, deep learning models have shown better results when the number of instances is higher and contain a true representation of the real world, since it will allow it to learn a wider range of features. Especially, in automatic object counting tasks, it is required to obtain a high variance of instances since it will allow obtaining features for elements with different sizes and orientations, as well as improving their recognition on unseen objects. Figure 12 shows a clear example of two samples from the CIFAR10 dataset of how different two elements from the same class can be.

Figure 12: High intra-class variance example from two *airplane* samples from the *CIFAR10* dataset.

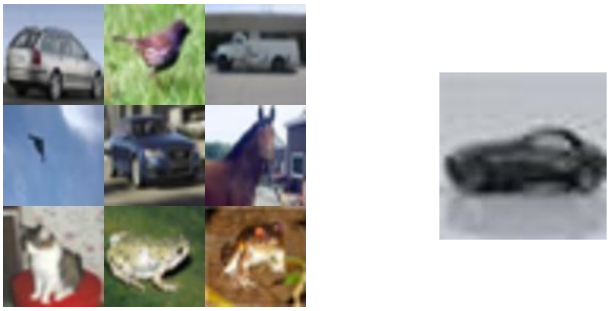
The following sections present the datasets that have been developed for the purpose of this task. These datasets have been defined with several configuration parameters to provide a higher adaptability to future works.

4.1 Count CIFAR10 Dataset

The Count CIFAR10 Dataset is a generated dataset defined only for this project. It uses sample images from the well known CIFAR10 dataset and sorts them randomly to form an image grid of the desired shape of the user. As a reminder, the CIFAR10 dataset contains 60000 32x32 colour images for 10 classes, with 6000 images per class. The classes are airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships and trucks. There is only one class element per image, therefore, there is no overlapping and all classes are mutually exclusive.

For this project, the defined shape was a 3x3 grid of images from different classes where the images have been rescaled to a size of 96x96 to obtain a higher resolution, as shown in Figure 13a. Additionally, it provides as input a template image for the desired class to be counted, as can be seen in Figure 13b. The desired output, and ground truth, corresponds to the number of instances in the image from the same class as the template. Therefore, this dataset approaches the counting task as a direct counting problem where the defined model is expected to find a correct mapping from the image grid and template to the desired scalar value. In the case shown in Figure 13, the output for

this sample would be the scalar value 2 since there are only two cars (don't confuse the segment on the upper right with a car since it is a truck).



(a) 3x3 Image grid. (b) Car template.

Figure 13: Input sample for the *Count CIFAR10 Dataset*.

The dataset is composed of 60000 288x288 colored image grids (due to the 3x3 defined shape and the 96x96 pixels from each original image) and 10 96x96 colored template images. Each time an instance is retrieved from the dataset, it provides, apart from the image grid, all ten templates and the count for each class in the picture. Therefore, the true size of the dataset is 60000 per each class (600000 instances).

Although the size of the dataset is pretty high, the reduced size of the input images allowed a fast training which was very helpful in the early stages of the project since the developed models could be easily evaluated on it. The original dataset created the image grids on-demand from random instances and the given input. Later, to fasten even more the training process, a separate dataset has been done by saving the created image grids and avoiding the time and resources required to perform the image transformations.

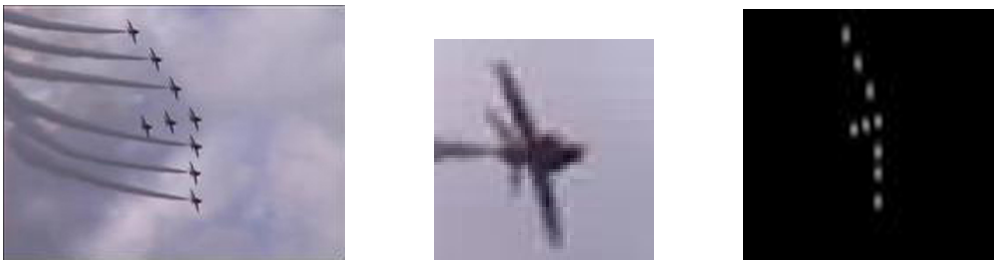
This dataset can be easily adapted to fit more classes, more instances, change the image resolution, or the shape of the grid using a set of configuration parameters. An easy way of increasing the number of classes and instances would be using as baseline the CIFAR100 dataset instead of CIFAR10. The developed implementation of the dataset is flexible enough to apply these modifications with minor changes.

The performance results for each direct counting model tested on this dataset will be shown in Section 6.1.

4.2 ILSVRC2015

The second dataset used in this project is a variation of the ILSVRC2015 dataset, extracted from the ImageNet annual challenge the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where different datasets are provided with the purpose of developing a software that is able to recognize the different objects in the data. These datasets go from object classification in images to object detection from videos.

This project uses frames and bounding box annotations from the 2015 ILSVRC challenge dataset for object detection in videos. However, slight transformations have been applied to make it fit in an object counting problem, more specifically, a spatial density counting problem. These modifications were based on the *Class-Agnostic Counting* paper [15] where the video frames are kept as is, but the annotations are used to define a Gaussian inside each bounding box. Additionally, the different bounding boxes are used to create the template image for discriminating the object to be counted. Figure 14 shows an example of an input image and template, as well as the output image with the Gaussians. The original input image is an input sample of the ILSVRC2015 dataset for object detection from videos, the input template and the ground truth image have been generated from the bounding box information provided by the dataset.



(a) Original input image. (b) Input template. (c) Ground truth.

Figure 14: Input and output sample from the ILSVRC2015 dataset

Since the dataset doesn't provide any information about the class that is being analyzed, the used templates are always part of the given image. This might cause a bias in the learnt features since they might be too specific, however, this problem is automatically fixed thanks to the fact that the dataset provides videos containing several elements from the same class and all of them are detected, *e.g.:* a pack of dogs with different hair colors.

The dataset contains 1952 training videos with different lengths and resolutions. Each frame has been carefully annotated with the information about the image properties and the information about the objects contained in it. The owners inform that due to the scale of the data, a minor subset of frames may contain errors in its annotations. Moreover, there are 30 different categories of objects in this dataset.

To overcome the problem of having images with different resolutions, the images have been resized to make them fit into any type of network. However, the resizing of images is an optional feature that can be turned off.

As can be seen in Figure 14c, the dataset provides as ground truth a one-channel image with a Gaussian at the location of the objects instead of providing a final number. Nevertheless, once the model is trained using this dataset, different algorithms can be used to count the local maximums from the output image.

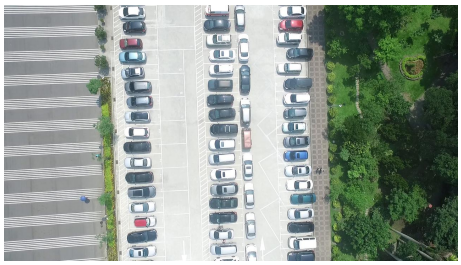
The performance results of models using this dataset will be described in Section 6.2.

4.3 CARPK

Finally, as briefly mentioned previously, a third dataset has been used for evaluating the degree of adaptability of the models to analyze data from unseen scenarios. This dataset is the CARPK dataset, which contains nearly 90.000 car samples in drone-captured images from parking lots, as shown in Figure 15a. It is an annotated dataset with labels for the bounding boxes of each car located in the image. The different cars are distributed throughout 989 training images and 459 test images, however, this project will only make use of the test data since the dataset will only be used for evaluation purposes. The images have been captured from different parking lots in order to avoid a learning bias and being able to detect cars in any environment.

In contrast with the CIFAR10, this dataset doesn't discriminate between trucks, vans or other types of cars. This fact will allow evaluating as well the degree of generalization of the trained models.

This dataset has been transformed in order to make it fit a spatial density counting task. As in the previous dataset, the ground truth has been transformed into a single channel image containing a set of Gaussian functions created inside the area of each bounding box annotation, as presented in Figure 15d.



(a) Original input image.

(b) Input template.

(c) Original Bounding boxes.

(d) Created ground truth.

Figure 15: Input and output sample from the CARPK dataset

Figures 15a and 15b, present a sample image from the dataset and a car template image to use as reference. Then, Figures 15c and 15d show the ground truth output provided by the original dataset as well as the generated one for this project. Additionally, the dataset provides a ground-truth count value for comparing the regressed count against it.

The performance results in this unseen dataset will be evaluated in Section 6.2.

5 | Methodology

This work approaches the counting problem from different angles with the purpose of analyzing which is the best procedure for achieving the count and finding a class-agnostic counting model capable of counting any object independently of the characteristics of the image.

This section will provide a detailed description of the deep learning models defined for facing the counting task in Section 5.1, followed by a description of the experiments and metrics used for their evaluation in Section 5.2.

5.1 Models

The architecture definition of the deep learning models developed for achieving a class-agnostic counting model has been based on some common architectures from the computer vision research area as well as on the analysis of how we, humans, perform object recognition.

The first concept which defined the structure of all future implemented models, was the decision of having two separated input images, one corresponding to the original image to count from and a second input with an example of the object to be counted. This decision was made after analyzing the counting problem from a human point of view. We (humans), in order to recognize a real object, we create a reference template after observing an initial example. From this example, we are capable of describing its characteristics like size, shape, extremities, etc. If we want our model to be able to count any type of object, it is necessary to first let it know the object to be counted. From that point, it will be its job to define the object's most relevant features and the ones that make it differ from other classes.

Once defined the input, we could proceed with the proposition of the desired output. Since the purpose of this project is analyzing and comparing the different approaches for object counting, both strategies (direct and spatial density counting) will be studied separately. As mentioned in Section 3, the output of the models designed for direct counting will be a single scalar value with the predicted count for the objects in the image. On the other hand, the output of the models following a spatial density counting approach will be a 2D representation of the density of the objects located in the image, as illustrated in Figures 14c and 15d.

Having said that, the defined models for approaching the object counting problem are the *Weight Sharing Network*, *Siamese ResNet*, *ETCNet* and *ETSCNN* for direct counting; and the *Generic Matching Network* [15], *Siamese GMN*, *GMN ETCNet* and *GMN ETSCNN* for spatial density counting.

	Learnable Parameters	Memory Size
Weight Sharing Network	26140929	104,56 MB
Siamese ResNet	26196801	104,78 MB
ETCNet	26196801	105,32 MB
ETSCNN	26196801	105,32 MB
Generic Matching Network [15]	6000897	24 MB
Siamese GMN	4477441	17,91 MB
GMN ETCNet	6000897	24,53 MB
GMN ETSCNN	4477441	18,44 MB
VAE	133363	533,45 KB

Table 1: Models parameter and memory size.

Table 1 presents the number of learnable parameters and memory size for each of the evaluated models. It must be said that in the case of the *ETCNet*, *ETSCNN*, *GMN ETCNet* and *GMN ETSCNN*,

we must add the number of parameters from the VAE because it is part of the model architecture, as will be explained in the following sections. However, since the VAE only has to be trained once and, after that, its weights are frozen, it was kept as a separate model in the table.

Moreover, we can observe how the models following a spatial density counting approach, contain a significantly smaller amount of learnable parameters. This is because all spatial density models are based on the *Generic Matching Network* [15] which, as will be explained in Section 5.1.5, uses a `ResNet-50` network architecture that is cut after the third block of layers, removing all the remaining parameters of the network.

Once the approach that each model will follow and its number of parameters is clear, let's proceed with the description of its internal architecture and how it works.

5.1.1 Weight Sharing Network

Based on the fact that the network must use the template's information to analyze the original image, the first idea that came to mind was using the same convolution filters to analyze the template like the ones to analyze the original image. With that purpose, the *Weight Sharing Network* was created.

The architecture of the *Weight Sharing Network* can be divided into two parts, the first part is formed by a Convolutional Neural Network in charge of creating the mapping between the convolution features and the count value. In order to improve its adaptability to new environments, a `ResNet-50` pre-trained with the ImageNet dataset has been used. Its input is the picture to be analyzed, as illustrated in Figure 16, but this part alone still lacks the information about the object to be counted. That information will be provided by the second component of the network, a convolutional layer that takes as input the template image and performs the convolution operation on it. The purpose of this component is to copy its weights into the first convolutional layer of the main network. In order to do that, the first convolution layer of the main CNN must have the same composition as the template's convolutional layer. The result from this convolution layer is not used for computing the final count, however, the layer can still be trained using the gradient that will be passed to it in the backpropagation step.

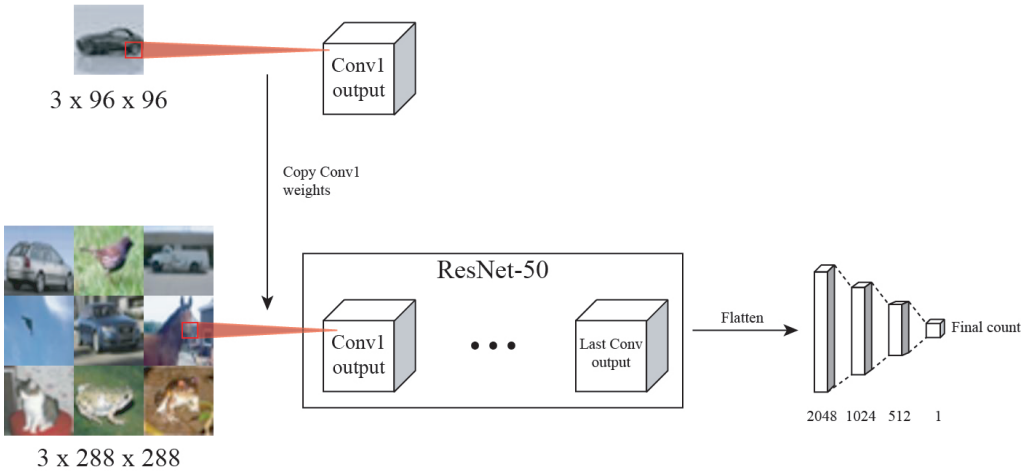


Figure 16: Functional architecture diagram of the *Weight Sharing Network*.

Figure 16 shows the architecture representation of the *Weight Sharing Network* implemented in this project where the main CNN is represented with a `ResNet-50` and the external convolutional layer is formed by 64 7x7 filters with a stride of 2 to fit the ResNet's first convolutional layer. Nonetheless, this network idea can be applied in any network architecture as long as it fulfills the requirement of having both convolutional layers for both inputs with the same size.

The internal forward process of this network is defined by the following steps:

1. First compute the convolution operation on the template image.
2. Then, the weights from the external convolution layer are copied to the first convolutional layer of the ResNet.
3. Once the weights are copied, the ResNet computes the count from the image to count from by performing the standard forward pass through the network until the last linear layer.

As can be seen, the behavior of the network is quite simple and only differs from a vanilla CNN when copying the weights from the external convolution layer. This simple model was defined with the purpose of doing a first step in the direction of achieving a class-agnostic direct counting model.

5.1.2 Siamese ResNet

As an expansion of the previous idea, the *Siamese ResNet* model has been developed with the goal of analyzing the performance of a verification network, like Siamese networks, on a counting problem.

As the name suggests, it has the layer architecture of a `ResNet-50` with the internal behaviour of a Siamese network. Its methodology is similar to the one used in object tracking problems with Siamese Convolutional networks but, instead of providing a template from a part of the original image, a different image from the class to be counted will be provided.

As mentioned previously, this model is used in a direct counting problem, therefore, the output value of the network is a scalar value with the final count for the object, as shown in Figure 17.

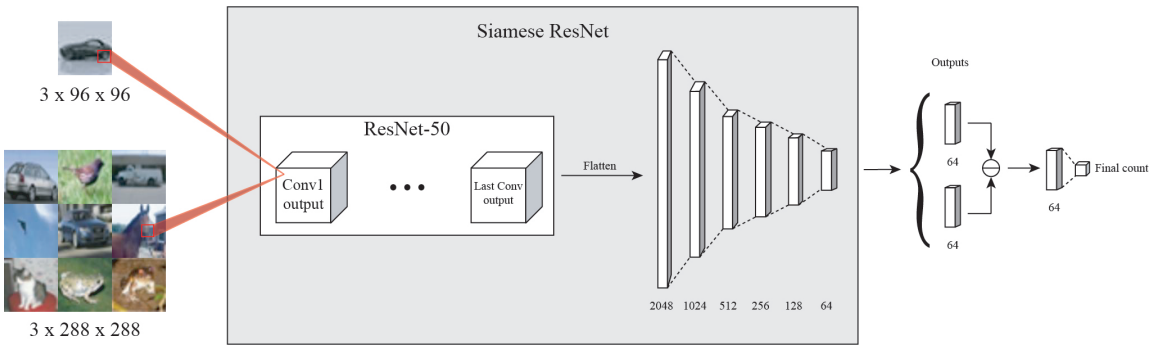


Figure 17: Functional architecture diagram of the *Siamese ResNet*.
The ⊖ operator represents the absolute difference between the computed vectors for each input.

One of the advantages of using a ResNet architecture for the convolutional layers is that, apart from having residual connections that help the model to adapt to different environments, it uses an adaptive average pooling layer right before the linear layers for being able to accept different sized images. This will be helpful since both inputs have different shapes. This fact could also affect the performance of the model if the difference between both shapes is too high which is a case that will be studied in Section 6.1.

The behavior of the *Siamese ResNet* is described by the following steps:

1. First the template image is passed through the Siamese ResNet computing its output vector.
2. After that, the same process is followed for the image to count from.
3. Once all output vectors are computed the channel-wise absolute difference is computed between them keeping only one vector with the same size.
4. Finally, this vector is passed through a final linear layer to compute the output count.

Step 3 states that the similarity operation used in this project for comparing the different output vectors is the absolute difference, however, any other similarity metric can be used.

5.1.3 ETCNet

The goal of the developed models is to achieve a class-agnostic counting model which is able to count any class without having to be trained or having a fast adaptation on unseen elements. The models described before can learn features on the trained instances but have difficulties to adapt to new classes due to the fact that the learnt features aren't based on the template but on the task and dataset.

When comparing it against the human process for object recognition, we learn to describe the objects that we see which helps its later recognition. Each object is described in a different way depending on its more obvious characteristics, like shape, size, etc. In our case, the input image must be analyzed in a different way depending on the given template. For example, if the template is a house it will look for features with straight lines and corners, however, if the input is a car, the wanted features will be rounder. Therefore, the learnt convolutional features must provide a description of the current object and not be biased by the supervised dataset.

With that goal in mind, a new model has been developed called the *Encoded Template Convolutional Network (ETCNet)*. Its main characteristic is the fact that it splits the counting tasks into two modules, the feature extraction and the counting model.

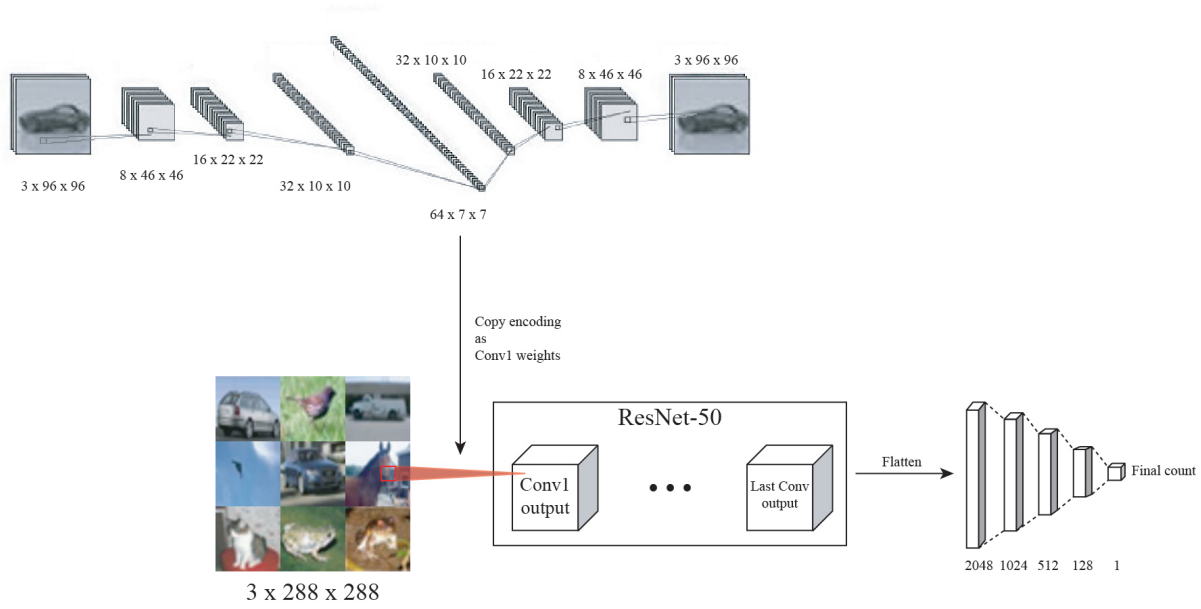


Figure 18: Functional architecture diagram of the *Encoded Template Convolutional Network (ETCNet)*.

Starting from the feature extraction, the main idea behind a feature extraction is to achieve a representation of the object in a lower-dimensional space. Since it is known that deep learning encoders are useful for that task, the feature extraction will be done with a convolutional Variational Autoencoder model. This model will compute an encoding of the template, with some semantic meaning, in order to obtain a smaller representation of it. In order to train it, each instance from the CIFAR10 dataset will be used as both input and ground truth for computing the loss of the decoded image. The encoded version of the template could be used as another feature to be added as input for the counting model, however that wouldn't be much different from directly using the original template, since the counting model would be able to extract the same information with the right amount of convolutional layers. Instead, the latent variables will be used as weights for the first convolutional layer of the counting model (thus the size of the latent variables must be the same as the shape of the convolution weights). These inferred weights will be frozen and the model will have to learn their relation with the original input image.

In order to fasten the forward computation during the training stage, all images from the same class, *i.e.* that use the same template, are passed in the same batch to avoid changing the encoded weights on each instance, therefore, optimizing the training stage.

The counting model, on the other hand, with the architecture of a `ResNet-50` is able to learn a mapping from the original image to the final count value with the help of the embedded features.

Figure 18 presents the internal architecture of the *ETCNet* and each of its components, inputs and outputs. The desired output value is achieved after the processing of both input images through the following process. These steps describe the forward pass of the network after the training of the Variational Autoencoder and loading its trained weights into the *ETCNet*.

1. The first step is defined by the encoding of the template's features and its embedding into the weights of the first convolutional layer of the counting model. As can be seen in Figure 18, the template image is encoded into a feature map of size 64x7x7. In order to make this encoding match the weights of the convolutional layer, which shape is 3x64x7x7 due to the RGB color channels, two different approaches are followed and compared in Section 6.1. The first approach is based on replicating the encoding three times and concatenating it. In contrast, the second approach is based on training three separate VAEs (one per color channel) and concatenating all three encodings. This latter approach increases considerably the number of learnable parameters of the model in exchange for a hypothetical better description of the template.

2. After copying the encoding as convolutional weights, the next step is computing the final count through the counting model as it would be done in any standard convolutional neural network.

This methodology differs from the previous models because, in this case, the model won't learn the features that better describe the object in the current dataset, instead, it will work from a description of the objects and try to find a relation between these features and the given image without any learning bias from the dataset. For this reason, it is important to keep separate the tasks of feature extraction and count retrieval.

It is important to remark that, in the ideal scenario, the more classes are fed into the Variational Encoder during the training stage, the more general will be the features and more flexible will it be to describe unseen objects. However, it is important to find the correct configuration for the latent space which will have to describe a wide range of different classes.

When training the model in unseen data, first one must see the performance of the pre-trained VAE and observe if it is able to describe this unseen object. In case it doesn't, it can be easily trained with a few instances without the need of having any supervised dataset, having a set of template examples will be enough. This allows an easier integration of the model to new environments.

5.1.4 ETSCNN

Finally, the last model used on direct counting is the *Encoded Template Siamese Convolutional Neural Network (ETSCNN)* a network that combines the architecture of *ETCNet* with the logic of the *Siamese ResNet*.

As in the *ETCNet*, it is composed of two modules: the feature extraction and the counting model. The network architecture for both models is exactly the same, however, the counting model applies the logic of a *Siamese ResNet*. Using as input both the template and the original image, with the inferred weights from the feature extraction model, it achieves a mapping between the ground truth count and both generic and specific features.

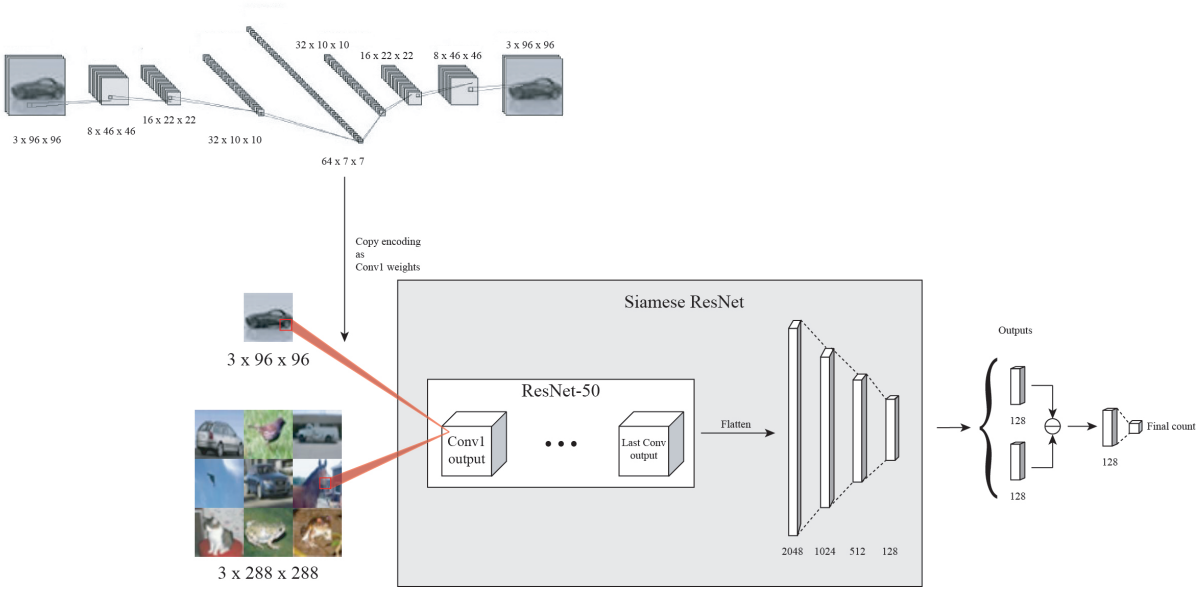


Figure 19: Functional architecture diagram of the *Encoded Template Siamese Convolutional Neural Network (ETSCNN)*.
The $\ominus$ operator represents the absolute difference between the computed vectors for each input.

The contributions of this model are the capability of using a general description of the object as well as a specific description based on the context of the dataset. This strategy might decrease the flexibility of the model on new classes due to the biased specific features and having to train the counting model as well, but having access to two types of features gives the model more information to learn from.

Once trained the Variational Autoencoder, this information is processed following these steps:

1. As in the previous model, first the encoding of the template is computed and copied as weights in the first convolutional layer of the counting model.
2. Next, the template is processed through the siamese counting model obtaining its output vector.
3. The same process is followed for the original image to count from, obtaining its output vector as well.
4. The similarity of both vectors is computed through a channel-wise absolute difference.
5. Lastly, the final count is regressed from the analysis of the similarity value through a linear layer.

This model has the advantage of being able to process a higher amount of information with the same number of learnable parameters as the *ETCNet*, as observable in Table 1.

5.1.5 Generic Matching Network

The first model in this project to work on a spatial density counting problem is the *Generic Matching Network (GMN)* from developed by Lu, E., et al. in their work on *Class-Agnostic Counting* [15]. As a summary, the network is composed of three modules, each module is named under their function in the network's behavior and their names are the *embedding*, *matching*, and *adapting* modules.

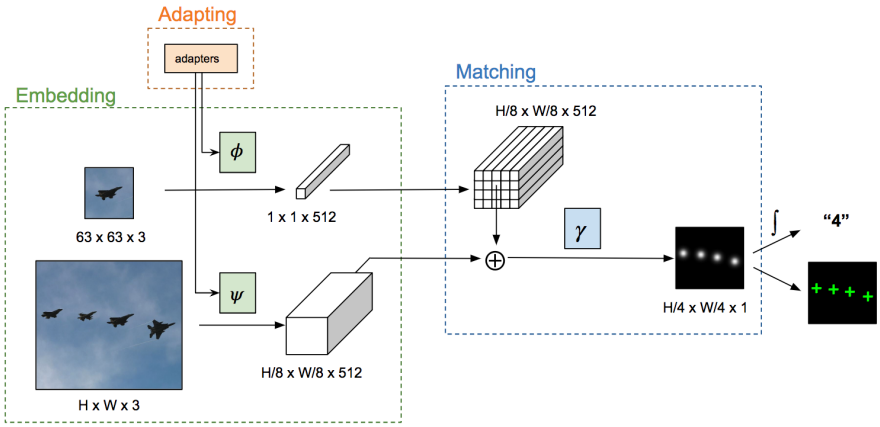


Figure 20: GMN module architecture extracted from the *Class-Agnostic Counting* [15] paper.

As observable in Figure 20, all three modules work together as one entity for obtaining the detected objects. The input to the network is the image to be analyzed along with a template of the object to be counted. This input is processed through the different modules of the network until reaching the final heat map with a Gaussian on the position of each located instance. From this point, the output can be returned as it is or be processed to return a scalar value by counting the local maximums on the map.

Once explained the general behavior of the network, let's describe the characteristics and functioning of each of the network's modules. Starting with the *embedding*, the task of this module is computing an encoding of both the full-resolution image and the template. The resulting encoding from each input has a different shape but same length. The template image is encoded into a feature vector while the full-resolution image is encoded into a dense feature map. The *adapter* module helps the *embedding* in the computation of an encoding by adding residual adapter modules [27] to provide an efficient domain adaptation. Finally, both encodings are used by the *matching* module for finding similarities between them. In order to compute the similarities through each region of the image, the template is compared against each feature vector of the dense feature map. With the purpose of optimizing the computation of the similarity, the template's feature vector is replicated until matching the same shape as the original image and then it is channel-wise concatenated to use it as a single input. The matching network is able to find the similarities between the input features and computes the output heat map.

The *embedding* module is formed by the inner structure of a cut `ResNet-50` after the second residual block and by adding extra residual connections for the *adapter* module. As in the previous models, we took advantage of the pre-trained weights of this network in the ImageNet dataset to fasten the training stage. On the other hand, the *matching* is composed of one 2D convolutional layer with 256 filters, a transpose convolutional layer for up-sampling the features and a final convolutional layer with only 1 filter in charge of building the output heatmap with only one channel.

This model will be used as baseline for comparing the spatial density object counting models during the experiments that will be mentioned in Section 5.2, as well as backbone for the architecture of the spatial density models developed in this project.

5.1.6 Siamese GMN

With the same architecture as the *Generic Matching Network*, a new network has been developed called the *Siamese GMN*. It reduces the number of learning parameters of the GMN, as shown in Table 1, by removing one of the two embedding networks and using the remaining one for encoding both the template and the original image.

When training the previous model (the *Generic Matching Network*), it was expected that the encodings from both inputs would be comparable and in similar scales. Nonetheless, this is only a supposition and it is led to the hands of the model to achieve this comparable representation, which requires more computational resources. The idea behind the *Siamese GMN* is to force the network to use the same network in both inputs, this way we ensure that the analyzed features from the input are the same, which eases the later matching process.

The remaining modules, the *adapter* and the *matching*, are kept as is in the parent model. Therefore the forward pass is as follows:

1. The first step is described by the encoding of the template image into a feature vector thanks to the *embedding* and *adapter* modules.
2. After that, the same process is followed on the full-resolution image obtaining a dense feature map.
3. Next, the feature vector extracted from the template is replicated to match the size of the dense feature map.
4. Then, both embeddings are channel-wise concatenated to be processed with the *matching* module.
5. Finally, the *matching* module performs the comparison and obtains the output density map.

5.1.7 GMN ETCNet

GMN ETCNet, as the name suggests, is a model based on the architecture of the *Generic Matching Network* and the *ETCNet*. Like in the model for the direct counting, mentioned in Section 5.1.3, the goal is avoiding the learning bias for the template features due to the supervised dataset. To prevent that from happening, a Variational Autoencoder, trained with the templates from the ILSVRC2015, will be used for learning the set of filters for the first convolutional layer of both networks inside the *embedding* module. The learnt filters will be set as weights on the layer and it will be frozen to avoid computing the gradient on it during the training stage.

The addition of the VAE creates a new module in the network in charge of preparing the network for the current input. For this reason, it was decided to name this block as the *prepare* module, as shown in Figure 21. It is expected that this new model will help the *embedding* to achieve a better description of the current input thanks to inferring which set of general features should be analyzed. This new module has been added under the hypothesis that it will help the model on adapting to unseen data thanks to providing a general description of the object which features were not biased by the counting task.

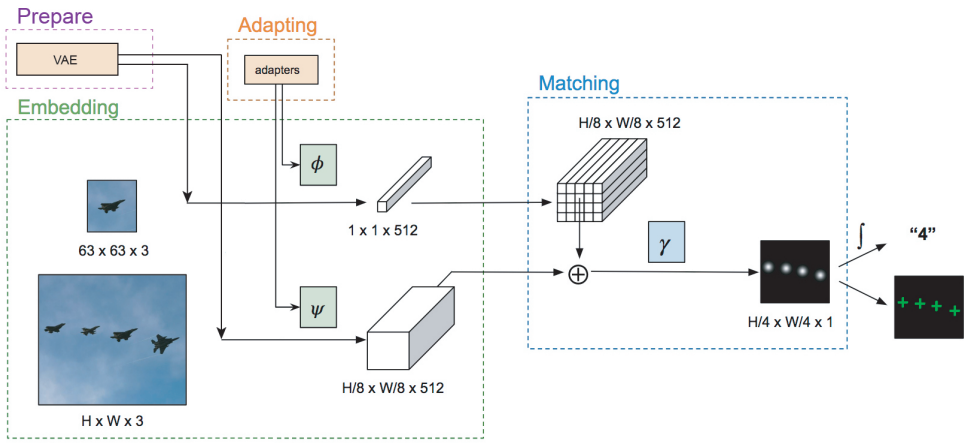


Figure 21: *GMN ETCNet* module architecture.

Unlike the *ETCNet*, in this network, it was not possible to optimize the forward pass since the classes of the objects are unknown in the dataset used for training the spatial density models. Because of that, each instance inside the batch is treated individually which slows down the training stage.

The forward process of this module is a mixture of the one from the *Generic Matching Network* and the *ETCNet*.

1. The model starts by computing the encoding of the template in order to use them as weights for the first convolutional layer of both *embedding* networks.
2. Followingly, both the template and the full-resolution image are transformed into a feature vector and a feature map respectively with each *embedding* network.
3. After that, the feature vector extracted from the template is replicated to match the size of the dense feature map.
4. Next, both embeddings are channel-wise concatenated to be processed with the *matching* module.
5. Finally, the *matching* module performs the comparison and obtains the output density map.

5.1.8 GMN ETSCNN

Lastly, the *GMN ETSCNN* model is the last network that will be analyzed for the task of spatial density counting. It is a variant of the *GMN ETCNet* where like in *Siamese GMN* both networks inside the *embedding* module are joint into one single network to analyze the same features on both images.

This model tries to take advantage of the benefits from the two previously mentioned models. First, it is able to achieve a more abstract and general description of the object to be counted and, additionally, it is able to reduce the number of learnable parameters since one of the embedding networks can be omitted.

Regarding the processing of the input images, the followed process is the same as in the *GMN ETCNet* with the difference that in step 2, the feature vector and the feature map will be computed from the same *embedding* network.

5.2 Experiments

This project evaluates the different ways of approaching the automatic counting task given an image and a reference of the object to be counted. With that goal, each of the previously mentioned models has been evaluated with their respective training dataset, mentioned in Section 4, based on their approach. Direct counting models are trained using the *Count CIFAR10* dataset, described in Section 4.1, and the spatial density models are trained on the *ILSVRC2015* dataset, explained in Section 4.2.

This section will provide a description of the training and definition process of the models, presented in Section 5.2.1. After that, Section 5.2.2 defines the experimental setup that has been followed for the analysis of the models proposed in this project.

5.2.1 Training process

This section will provide a brief reminder of what it means to train a convolutional neural network, as well as the decisions that have been taken through the different training iterations of the model until reaching their final version. Nonetheless, this section will not get into the details of how backpropagation and gradient descent works. Instead, it will describe the different components that define the training of a deep learning model and how they affect the performance of the model.

Training configuration

In order to train a deep learning model, one must first define a set of hyperparameters. These hyperparameters define how the loss will be computed from the supervised data, how much should this loss affect the weights of the network and for how long will the model be trained.

Starting with the loss functions, they define how the error between the prediction of the model and the ground truth value will be computed. Since the datasets used to train deep learning models are too big to be stored in memory, it is divided into smaller groups called mini-batches and the loss is computed for each one of them. The different loss values computed for each mini-batch will iteratively update the weights of the network thanks to the backpropagation and stochastic gradient descent algorithms.

There are several metrics that can be followed depending on the task to be done, therefore, depending on the output of the network. In the case of this project, since the output from both approaches are scalar values, the selected loss function for training the model is the Mean Squared Error (MSE), presented in Equation 5.1, where y_i is the ground truth and $\hat{y}_i$ is the prediction from the model. Other alternative losses like the Cross-Entropy Loss are not the best option since they are thought to be used on categorical output values.

$$MSE = \frac{1}{N}\sum_{N}^{i=1}(y_i - \hat{y}_i)^2 \tag{5.1}$$

Once a loss function is selected, the model designer must choose which optimization algorithm, also known as optimizer, will be used. An optimization algorithm helps the neural network to learn faster by deciding how to update the weight of the network and by modifying its learning rate depending on the current state of the network. Several optimization algorithms also use a component called Momentum. The momentum of an optimizer is a technique that accumulates the computed gradient for the past iterations of the network and updates the weights of the network not only based on the current loss but the direction of all the previous gradients.

In contrast with the loss function, there are multiple valid options when selecting the optimization algorithm that will update the model state. A whole analysis could be performed for selecting the optimizer that better fits each model. However, since it is out of the scope of the project, all the models in this project have been evaluated with the same optimization function, the Adam optimizer. Different studies [26], have shown great flexibility on several tasks from the Adam optimizer thanks to its momentum algorithm and its bias-correction. Because of that, this optimizer has become the most common choice among researchers.

The last component to be defined is the maximum number of epochs. As previously mentioned, the datasets in deep learning are usually divided into a group of mini-batches. An epoch is when the loss for all the mini-batches of the dataset has been computed. Computing the loss from the whole dataset is not enough for the model to be able to learn the necessary features that are required to complete the task. For this reason, several epochs are passed to the model until it reaches a convergence point where the loss stops decreasing.

The models have been trained for 100 epochs maximum. Nonetheless, in case the convergence point has been reached before the 100 epochs, the training process has been stopped since the model will hardly improve its performance from this point onward. Moreover, in order to save the model with better performance values, Early stopping regularization has been applied to prevent the model from worsening its validation loss even if the loss starts increasing as the epochs pass by.

Due to the high dimensionality of the *ILSVRC2015* dataset, the models following a spatial density counting approach were trained through several iterations thanks to saving a checkpoint at each epoch. The checkpoint contains the information about the model architecture, its weights, the current epoch and the current state of the optimizer. This allows stopping and resuming the training of the models to be evaluated.

Table 2 summarizes the hyperparameters that define the training process of the evaluated models. Once the training configuration has been defined, each model has been analyzed and modified based on their performance results. The following sub-section will describe the decision process followed to achieve the final version of the models presented in Section 5.1.

Loss function	MSE loss
Optimizer	Adam with learning rate $1e^{-3}$.
Max. epochs	100

Table 2: Training hyperparameters configuration.

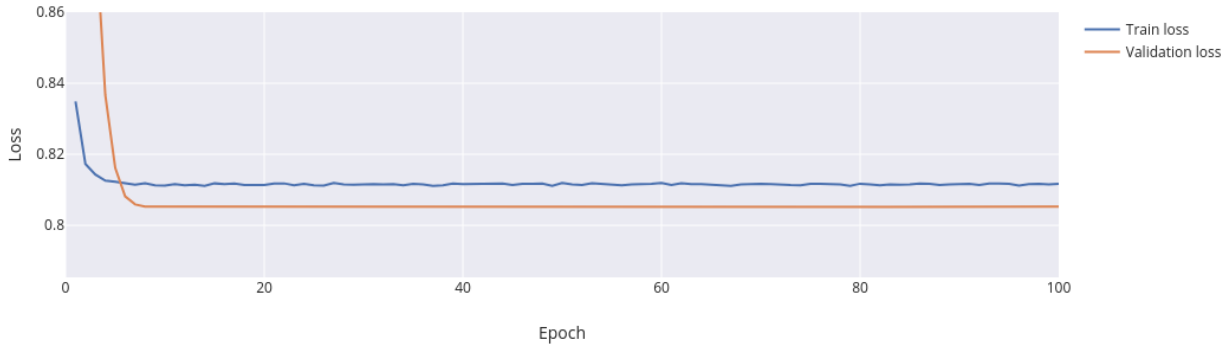
Performance analysis

The goal of a deep learning model is to be able to learn a set of features that allows it to generalize and analyze any kind of input inside the scope of the model, including unseen instances.

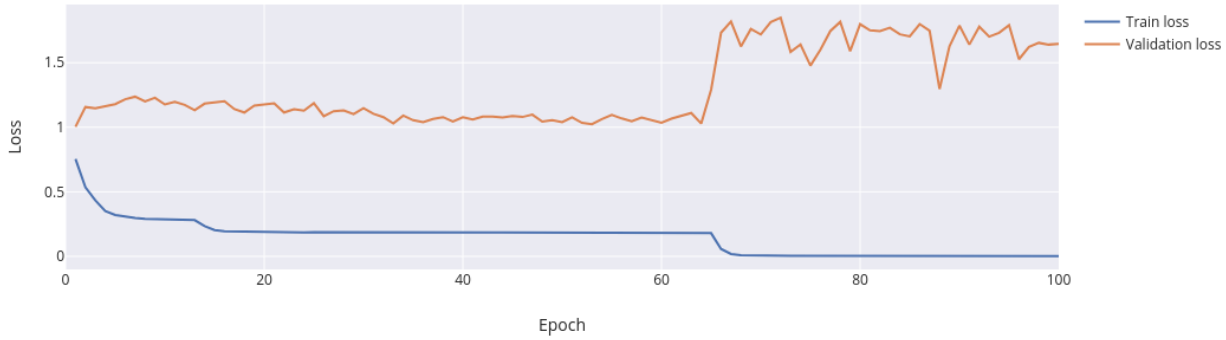
In order to be able to evaluate the generalization of the developed model, one must divide the dataset into separate groups with a set of instances that the model will learn from and another set with unseen instances. These sets of instances are commonly separated into three data splits: the training set which will be used for learning the patterns in the data, the validation set containing unseen instances to be evaluated during the training process and the test set with instances that are not contained in either of the previous sets to be used for evaluating the model after being trained.

However, achieving a model that is able to learn features that describe all three data splits is not a trivial task. For the sake of analyzing if the model is correctly defined to finds these kinds of features, one must observe the computed loss value for both the training and validation sets. These losses are stored at the end of each epoch for evaluation purposes and can then be plotted in a graph as what is know as learning curves.

The two main issues that make generalization a hard task are Overfitting and Underfitting. Overfitting is when the model learns a set of features that describe the data contained in the training set but are not able to generalize to the other sets. This problem can be detected when there is a considerable gap between the training and validation curves, as shown in Figure 22b.



(a) Underfitting example.



(b) Overfitting example.

Figure 22: Visualization of Underfitting and Overfitting through the learning curves of the model.

A model showing overfitting indicates that the model is so complex that it is able to learn the details and noise of the training data which affects its performance on the evaluation data. There are several techniques for fixing overfitting. Among which the most common ones are:

- Normalizing the data.
- Reducing the complexity of the network (by removing layers or reducing the number of neurons per layer).
- Adding dropout layers and batch normalization layers inside the network.
- Applying regularization to prevent the model from having large weights.

On the other hand, Underfitting is when the model is not able to learn the patterns in the training set due to having an architecture that is too simple or that doesn't fit the current task. Underfitting can be detected when the learning curve doesn't show any decrease in their loss, when the validation loss is below the training loss or when neither of the learning curves achieves an acceptable loss value for the current task. The easiest way of fixing underfitting is by increasing the complexity of the network or by increasing the amount of data available (if possible).

In order to achieve the models presented in Section 5.1, this performance analysis has been done for each model applying the necessary modifications to achieve a model with the lowest underfit and overfit as possible. Once all the models have been evaluated and validated, we could proceed with the real experiments from this project for the evaluation of their performance in the task of class-agnostic counting.

5.2.2 Experimental setup

Each model will be evaluated once with their respective dataset with only three exceptions where multiple input configurations or network architectures will be analyzed. The results from each experiment will be provided in Section 6, where both counting methodologies are analyzed finding their strengths and weaknesses. Moreover, each model has been evaluated individually with the purpose of proposing a new Class-agnostic counting model.

The three exceptions are the *Siamese ResNet*, the *ETCNet* and the *GMN ETCNet* models. For the sake of obtaining a better understanding of the defined models and trying to achieve the optimal configuration for each model, different scenarios have been analyzed for these three models.

Starting with the *Siamese ResNet*, it is known that ResNet models use adaptive pooling in the last convolution layer of the network to accept different input shapes, however, it is possible that it affects the performance of the model if the difference is too high. Because of that, three scenarios were analyzed by applying different transformations on the template input image. The three scenarios to analyze are:

- Add padding on the template to make it fit the original input size.
- Resize the template to the size of the original input
- Don't apply any transformation and keeping the raw template as it is.

The investigation of these scenarios will allow studying the degree of adaptability of the network on images of different sizes and different resolutions of the features to analyze. In other words, even if the template is not in the same scale as the given image to count from, the model is still able to find the references of the features presented in the template on the original image.

For the second exception, two model architectures have been evaluated for the *ETCNet* with the purpose of analyzing if encoding each color channel separately obtains a better performance of the model. With that goal, four Variational Autoencoders have been trained, one per color channel (Red, Green and Blue) and one grouping all three channels. Having three different encodings allows obtaining more detail on the description of the template, however, it has the drawback of having to train three separate VAE increasing the number of learnable parameters.

Finally, the last exception uses the *GMN ETCNet* to evaluate the performance of the model using two different VAEs for encoding the template. The performed experiments with the *ETCNet* and *ETSCNN* for direct counting, have used a VAE which has been trained under the CIFAR10 dataset. Nonetheless, a second Variational Autoencoder has been trained under the ILSVRC2015 dataset for being used on the models following a spatial density counting approach. In order to compare the encoding from both VAEs, two runs will be done with the *GMN ETCNet* using each of the Convolutional Variational Autoencoders.

Once specified the exceptions, the rest of the models have been evaluated with the input structure and network architecture described in Section 5.1. Moreover, since the experiments will start with the analysis of the direct counting models and then the ones with spatial density output, the evaluated scenarios for the *Siamese ResNet* and the *ETCNet* will only be performed in their direct counting variant under the assumption that the same conclusions will be extracted from the spatial density alternative.

Finally, after all the experiments from each counting strategy have been performed, the models will be evaluated on a test set and compared between them using the metrics described in the following section.

5.2.3 Metrics

As previously mentioned, Section 6 will evaluate each model by comparing their learning curves. Later on, once all models have been trained, a final comparison of the models will be performed under the test set. In order to compare the different models and approaches, the evaluation metrics for all experiments must be on the same scale. For this reason, the loss function used in all learning curves is the Mean Squared Error (MSE) loss between the ground-truth value and the prediction. From these curves, we can see how fast a model adapts to a new environment, as well as the minimum loss value.

Since the output is different for each counting strategy, evaluating their learning curves is not a valid option. Because of that, the comparison of different counting strategies will be based on the Mean Squared Error, the Mean Absolute Error and the accuracy on the test set. Additionally, in the case of spatial density models, the resemblance between the predicted output, its ground truth and the original image will be also evaluated.

6 | Results

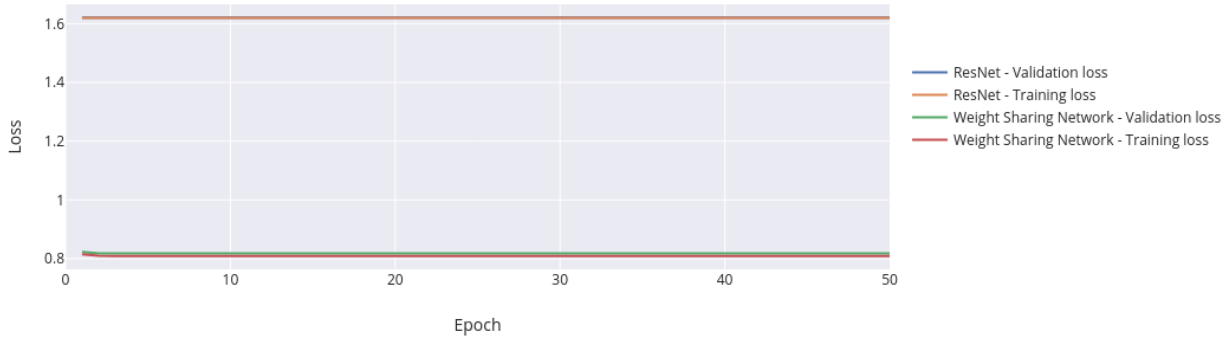
As discussed in the previous chapter, this section presents the analysis of the results of the conducted experiments. The experiments are divided into two groups based on their counting strategy. Section 6.1 presents the performance results from each defined direct counting model. Next, in Section 6.2, the performance results of the spatial density experiments are analyzed. Finally, Section 6.3 compares the benefits of each approach based on the results observed in the previously mentioned sections. Additionally, a final model is presented as the new step in the direction to achieve a true class-agnostic counting model.

Nonetheless, in order to analyze more clearly the learning curves, not all the epochs will be shown in the following sections, instead, the figures showing the performance of each model will be cut on the convergence point where the performance of the models doesn't improve in the remaining epochs.

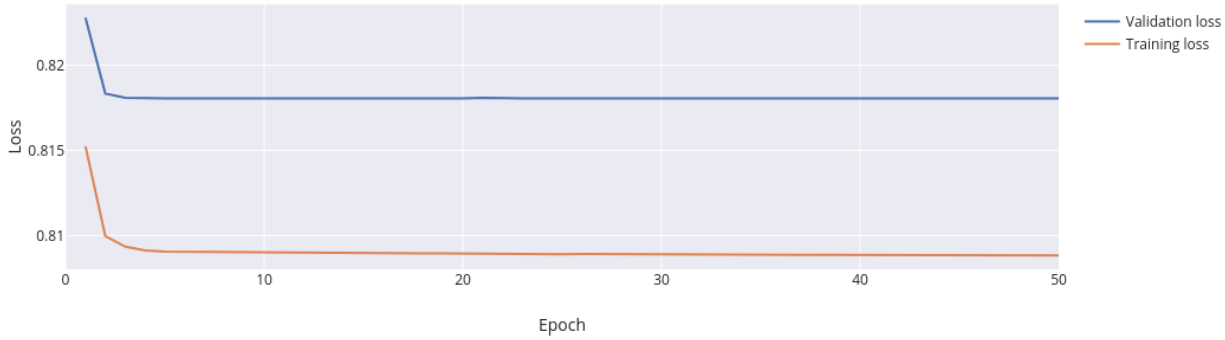
6.1 Direct counting experiments

All direct counting experiments were compared against the validation loss curve of training a vanilla `ResNet-50`, pre-trained with the ImageNet dataset, and adding 3 fully-connected linear layers with 1024, 512 and 1 neurons respectively to fit the current task. ResNet models have shown a great adaptation to unseen environments thanks to their residual connections. For this reason, this type of network architecture has been used as baseline for the comparison of the direct counting models. As briefly mentioned in previous sections, all direct counting models have been evaluated on the *Count CIFAR10 Dataset* but this model has only used as input the image grid. Because of that, the performance of this baseline model on this dataset wasn't good enough to obtain a good prediction for the count due to its lack of reference for the object to be counted, which made it predict always the same count value.

Once obtained the baseline performance, the defined models for this counting strategy were evaluated. Starting with the *Weight Sharing Network*, the loss obtained on the first epoch was already smaller than the baseline model, as observable in Figure 23a. This is due to the fact that the vanilla ResNet, was fed with just the image to count from, without any information about which element is wanted to be counted. Instead, in this case, the template is fed into the network along with the image to be analyzed. The second aspect to remark is the fact that, unlike the baseline model, the *Weight Sharing Network* presented a learning curve decreasing considerably its loss during the first 5 epochs, as can be seen in Figure 23b. That fact meant that the model was able to learn something from the dataset. Nonetheless, from that point onward, both training and validation loss got stuck and almost didn't change even if epochs passed. This proved that the model had a limitation to be able to learn the mapping between the input and the desired count. Because of that, the *Siamese ResNet* model was developed in order to see if the Siamese architecture could help in finding the relations between both inputs.



(a) Performance comparison of the `ResNet-50` against the *Weight Sharing Network*.

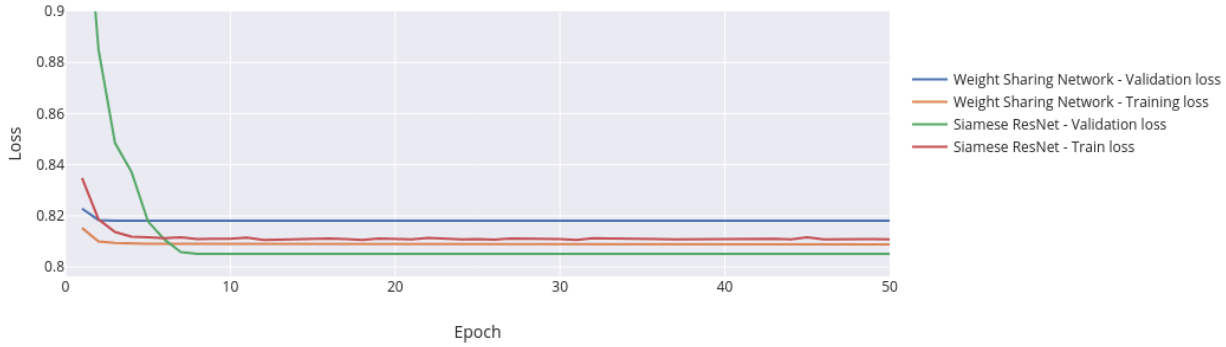


(b) Training and validation loss curves of the *Weight Sharing Network*.

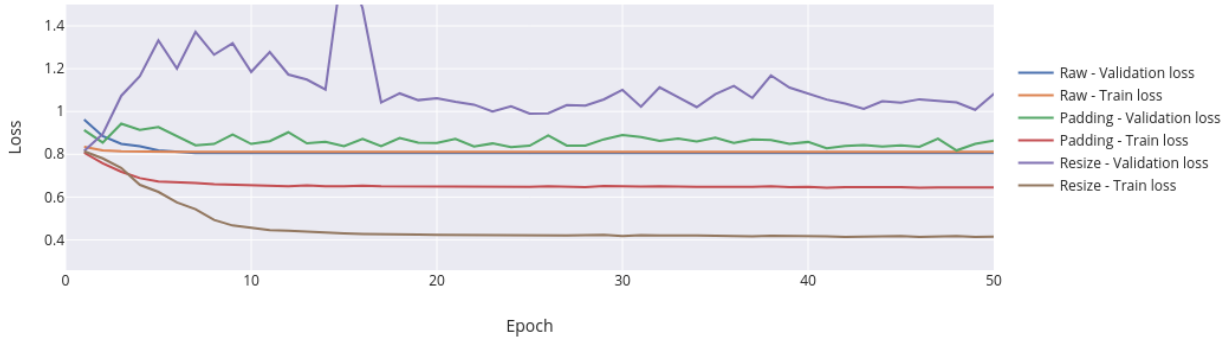
Figure 23: Learning curves for the experiments with the *Weight Sharing Network* with a Mean Squared Error criterion.

The analysis of the *Siamese ResNet* has been performed through several experiments to obtain the most efficient input configuration, as mentioned in the previous section. After obtaining the learning curves for all configurations, they could be compared for concluding which approach is the best. On the one hand, the *Siamese ResNet* experiments applying transformations on the template to make it have the same size as the original input (by applying padding and resizing the image) achieved a lower training loss but higher validation loss than both the siamese model with raw input and the previously evaluated model, as shown in Figure 24b. In other words, the models presented a slight overfitting even after applying diverse regularization techniques. Therefore, it proves that keeping the true scale of the template improves the performance of the model. However, it also proves that having the template on a different scale with respect to the original image input makes the training harder but the model is still able to achieve a mapping between the input images and the final count.

On the other hand, training the network with the raw images (without any transformation) obtained an underfitted model with a higher training loss but lower validation. When trying to fix this underfitting, the validation loss obtained was higher than the one obtained from the underfitted model, for this reason, and for achieving a lower validation loss than the *Weight Sharing Network*, as illustrated in Figure 24a, it was decided to keep it as the best model up until this point even though it was clearly underfitted.



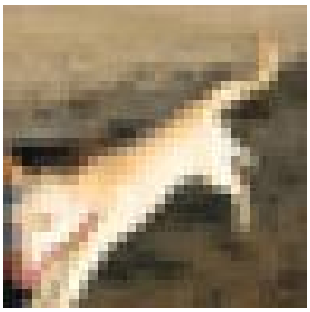
(a) Performance comparison between the *Weight Sharing Network* and the *Siamese ResNet* during the training stage.



(b) Training and validation loss curves for the evaluated configurations on the *Siamese ResNet*.

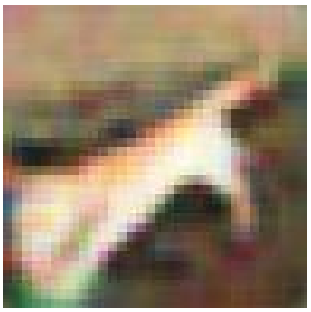
Figure 24: Learning curves for the experiments with the *Siamese ResNet* with a MSE criterion.

Following up, the next model to be evaluated is the *Encoded Template Convolutional Network (ETCNet)*. As mentioned in the definition of the model, this architecture tries to separate the feature extraction from the counting task. For this reason, we must ensure first the correct functioning of the feature extraction before starting with the counting task. In order to do that, we will use the validation data and compare the original image against its decoded version. Like in the previous model, two different approaches were evaluated for this model achieving an encoding per color channel or only one encoding. Figure 25 presents the comparison of the decoding from both approaches against the original image. In both cases, the decoding is good enough, taking into account that Variational Autoencoders usually achieve a blurred version of the original image due to the lack of knowledge about how real images are. This could be improved by using Generative Adversarial Networks or Discriminative network models to achieve more realistic representation. Nevertheless, achieving a realistic representation of the decoded image is not the purpose of the model. For this reason, and based on the similarity of the decoded images, both VAEs were validated for using them in the *ETCNet* and *ETSCNN* models.



(a) Original template.

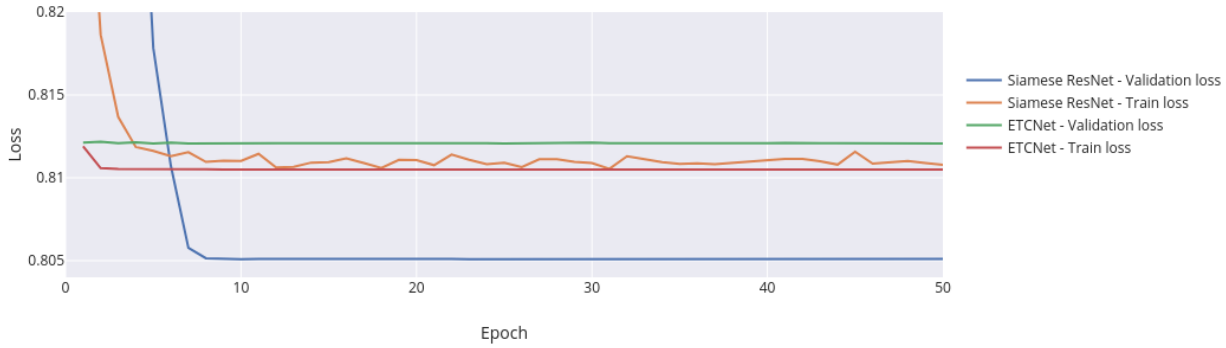
(b) Single template decoding.



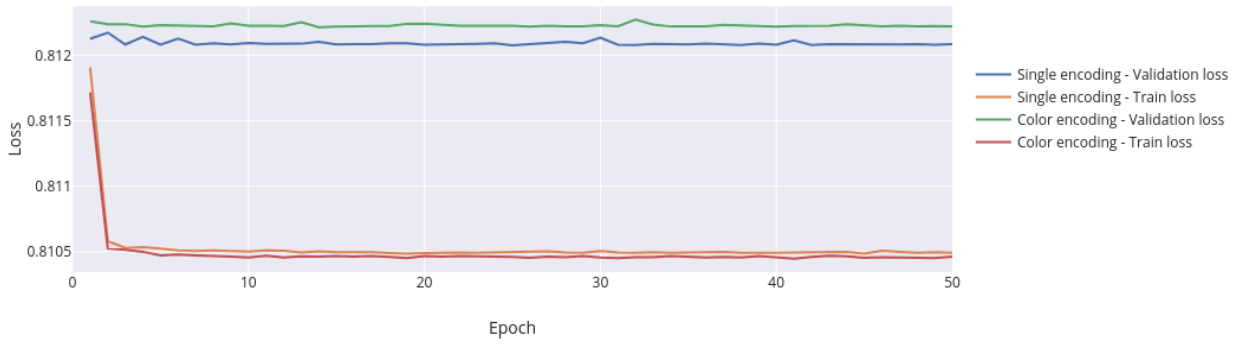
(c) Channel template decoding.

Figure 25: Template decoding through a Convolutional VAE of an image from the CIFAR10 dataset.

Regarding the counting task for the *Encoded Template Convolutional Network*, as already mentioned in the previous section, the hypothesis was that providing an encoding for each channel would obtain a better performance in exchange for slower training. Nonetheless, the obtained decoding, illustrated in Figure 25c, showed a slightly worse performance for the alternative encoding each channel individually. Figure 26b presents the learning losses of both approaches and, as can be seen, the model with a single encoding obtains a slightly better validation loss proving that it is not worth it to train each channel separately. When comparing both approaches with the previously evaluated model in Figure 26a, it can be observed how the validation loss is higher than the previous model, the *Siamese ResNet*. This fact was the precursor of the creation of the *Encoded Template Siamese Convolutional Neural Network (ETSCNN)*.



(a) Performance comparison between the *Siamese ResNet* and the *ETCNet* during the training stage.



(b) Training and validation curves for both encoding configurations evaluated on the *ETCNet*.

Figure 26: Learning curves for the experiments with the *ETCNet* with a MSE criterion.

Finally, the last evaluated model for the direct counting experiments is the *Encoded Template Siamese Convolutional Neural Network (ETSCNN)* which aims to aggregate the benefits from both previous architectures into a single model. Unfortunately, the learning curve of this model was pretty similar to the *ETCNet* model, therefore, there is no benefit from applying this model.

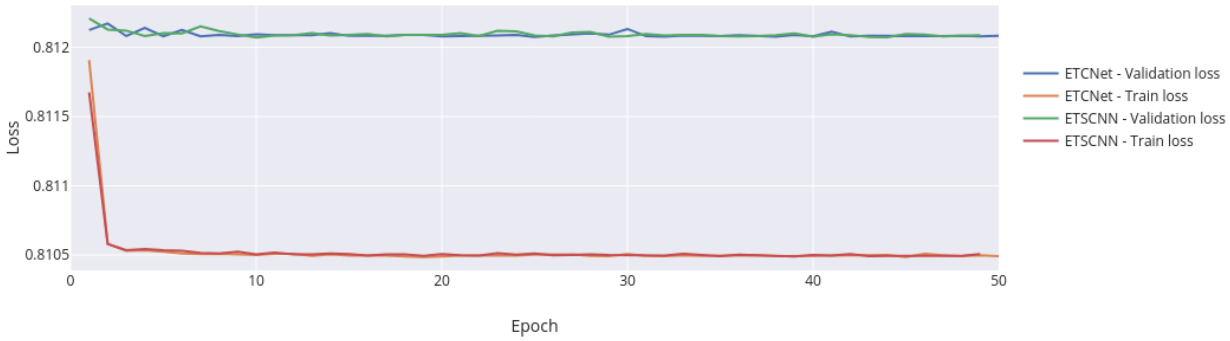


Figure 27: Performance comparison of the *ETCNet* against the *ETSCNN* through their learning curves.

After all models from this approach have been trained and evaluated, we can conclude that the best tested direct counting model is the *Siamese ResNet* which obtained the lowest validation loss curve for the tested models. However, the learning curve of a model is not enough for validating its efficiency. For evaluating the model's efficiency, the mean squared error(MSE), mean absolute error (MAE) and the accuracy have been computed for each model with the test data split, each metric is presented in Table 3. By observing this table, one can compare the performance results from each model based on their metrics. As can be seen, all models (except the baseline model) achieve similar scores, with the exception of the models using a Siamese behavior which achieved a slightly lower Mean Absolute Error. In the case of the ResNet model used as baseline comparison, after analyzing its predicted values we observed how it always predicted the value 0, showing no learning at all as presented in Figure 23a. This explains having a higher MSE and MAE loss but a higher accuracy since the dataset might contain a higher number of negative instances. For the other network models, the predicted output is not constant but still, its prediction is still far from being useful and reliable. This fact proves that the models are underfitted and are not able to learn the necessary mapping from the image to the final count.

	MSE Loss	MAE Loss	Accuracy
ResNet	1.6245	0.9	38.8951%
Weight Sharing Network	0.8145	0.7003	38.4773%
Siamese ResNet	0.8145	0.6998	38.4773%
ETCNet	0.8145	0.7003	38.4773%
ETSCNN	0.8145	0.6999	38.4773%

Table 3: Performance result of the direct counting models on the test set.

Since the results from all models are not good enough, this makes us think that using deep learning models for class-agnostic direct counting isn't probably the best approach due to how difficult is for the network to find a mapping between the input images and the output scalar value. To prove that, the models developed for the spatial density counting approach have a similar architecture to the previously evaluated models.

6.2 Spatial density experiments

Unlike the previous experiments, the spatial density models were compared against the validation loss curve of training a *Generic Matching Network* on the *ILSVRC2015* dataset. This decision has been made since this model is thought to be the current class-agnostic counting model with the best performance results. Additionally, all the defined spatial density models are variations of the GMN in order to fit better this counting strategy.

Moreover, due to the high dimensionality of the *ILSVRC2015* dataset, as well as the time and resources limitations from this project, not all the spatial density experiments have been performed in 100 epochs. Instead, once the learning curves from each model showed the signs of arriving at a convergence point, the model was no further evaluated, keeping that checkpoint as the best model. Nonetheless, in order to keep a fair comparison for the models, all models will be stopped at the same epoch evaluating both their minimum loss and their training speed.

As in the previous section, this section will first analyze the different learning curves of each model and then proceed with a comparison of their performance metrics on the test set, including also in this case a comparison of their predicted outputs. After that, as previously described, a final evaluation on the CARPK dataset will be performed in Section 6.2.1 to evaluate the adaptability of the models on unseen data.

Starting with the *Siamese GMN*, the goal is to evaluate if the behavior of Siamese networks could compensate for the work that the weights of the second embedding network were doing. In fact, in order to compare the encoding from both images, both networks would probably have similar weights in both networks to obtain comparable encodings from both images. Once trained the *Generic Matching Network* and the *Siamese GMN* in the *ILSVRC2015* dataset, our hypothesis could be proven by observing how the learning curves from the *Siamese GMN* got similar loss values than the baseline model, as can be seen in Figure 28. The training loss shows a slight improvement but the validation loss is pretty similar to the GMN (except for the loss spike in epoch 14). Even with how similar both validation losses are, the *Siamese GMN* is a better option due to having less learnable parameters, as shown in Table 1, which fastens the training and prediction processes. Besides, using the same network ensures having both encodings in the same scale which eases the *matching* process as mentioned in previous sections. Therefore, this experiment was a success since it achieved the performance of the baseline model with less amount of parameters.

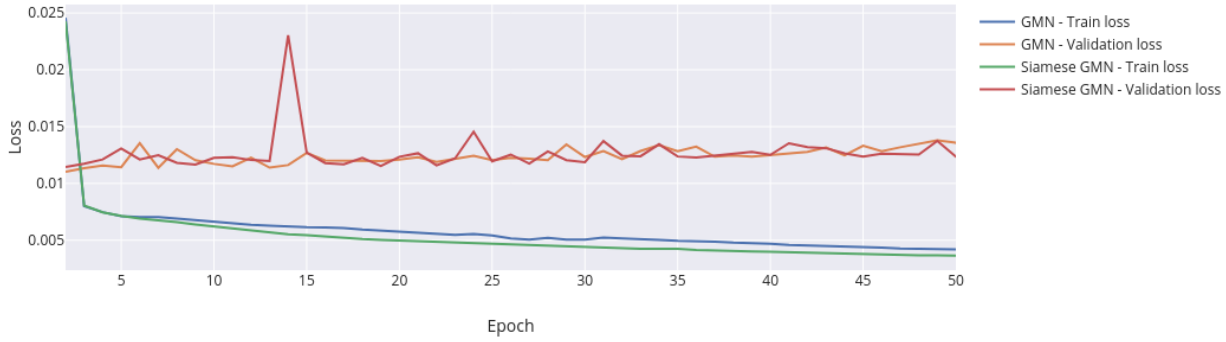
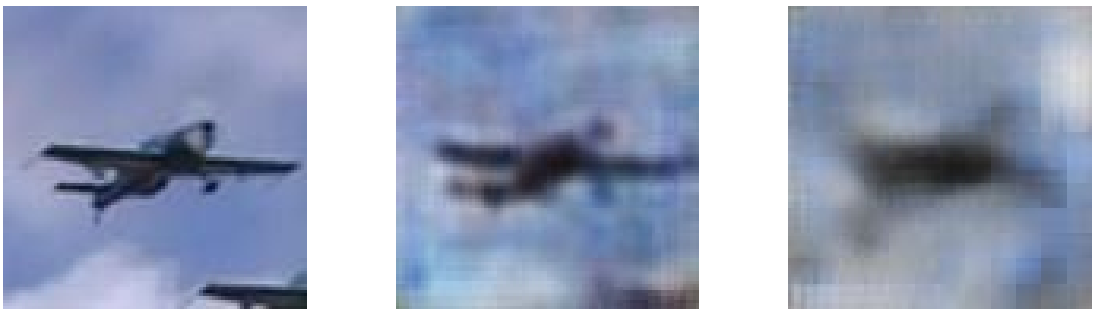


Figure 28: Performance comparison of the *GMN* against the *Siamese GMN* through their learning curves.

Following up, the next step was the analysis of the *GMN ETCNet* model. In this case, as mentioned in Section 5.2.2, two separate scenarios were evaluated for studying if the trained Variational Autoencoder from the CIFAR10 dataset was able to encode the templates presented in the

ILSVRC2015 dataset. With that purpose, a new Variational Autoencoder has been trained with the ILSVRC2015 to see if the performance of the previous model was comparable to the one obtained by a VAE specialized in the current dataset.

Surprisingly, after observing the performance of both trained VAEs on a template from the ILSVRC2015 dataset, the decoded template from the CIFAR10 VAE presented a higher level of detail in the image, as shown in Figure 29b. The reason behind that worse decoding is because the ILSVRC2015 has a higher number of classes which makes its learning process harder. Even with that, since the relevant part is the encoding of the template, both autoencoders have been used for training two separate *GMN ETCNet* models and evaluate the best approach based on their counting performance.



(a) Original template. (b) CIFAR VAE decoding. (c) ILSVRC VAE decoding.

Figure 29: Template decoding comparison of Convolutional VAE trained with different datasets.

Once both *GMN ETCNet* models have been trained, their learning curves were evaluated to see their difference in how they affected the model. As can be seen in Figure 30, both strategies obtained similar loss values with a slight improvement for the VAE trained with the ILSVRC2015 dataset. Seeing how little the improvement was, made us think that it is not worth training the Variational Autoencoder on a dataset as big as the one from the ImageNet challenge for almost no change in the model performance. Therefore, the approach that was kept as the best configuration was the *GMN ETCNet* with a Variational Autoencoder trained with the CIFAR10 dataset due to its simplicity and fast training. However, since it was still unclear which configuration would work better on unseen data, both approaches have been evaluated on the CARPK dataset as will be seen in Section 6.2.1.

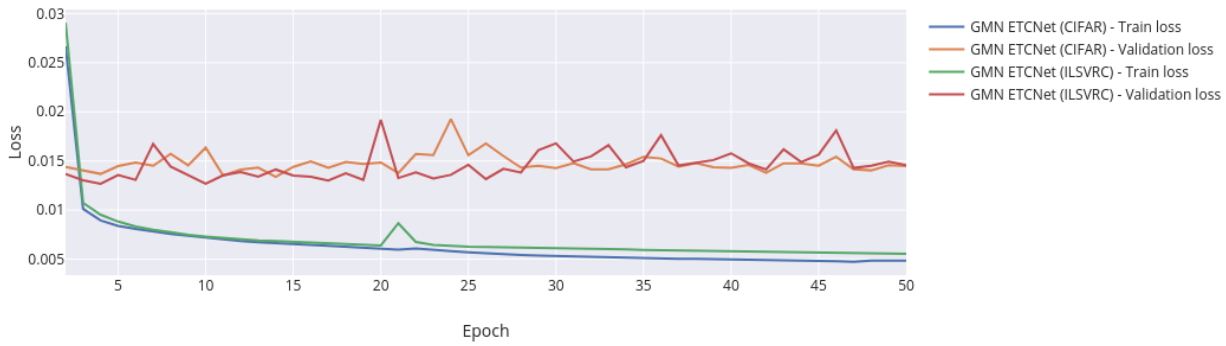


Figure 30: VAE configuration analysis through their learning curves for the experiments with the *GMN ETCNet*.

Thence, after observing how the CIFAR10 VAE was able to generalize the features for unseen

data, both models were compared against the previously evaluated model, the *Siamese GMN*, which has been decided to be the best class-agnostic spatial density counting model up to this point in the experimental process. In Figure 31, we can observe how both *GMN ETCNet* approaches, although they achieved great loss, the siamese model was still better in what refers to validation and training loss values.

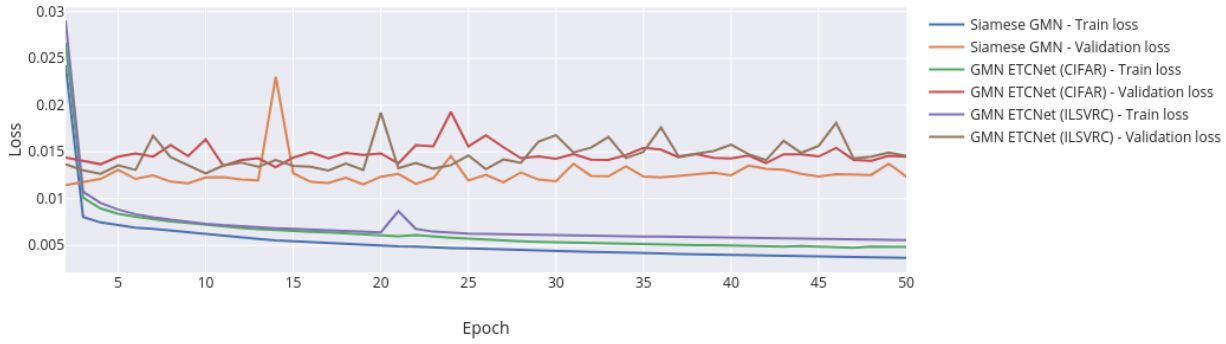


Figure 31: Performance comparison of the *Siamese GMN* against the *GMN ETCNet* configurations through their learning curves.

Finally, the last trained model was the *GMN ETSCNN* a model combining both siamese and the encoded template approaches trying to obtain the benefits from both. This model was evaluated by using the Variational Autoencoder trained with the ILSVRC2015 as the *prepare* module. The learning curves obtained when training this model are similar to the *GMN ETCNet* and, when compared against the *Siamese GMN* as in Figure 32, we can see how the values are good but not as low as the siamese model.

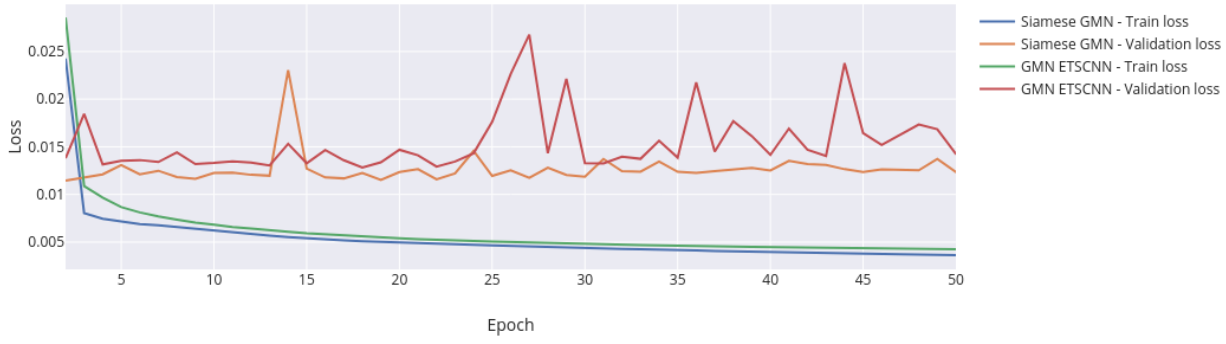


Figure 32: Performance comparison of the *Siamese GMN* against the *GMN ETSCNN* through their learning curves.

At this point, once all models were evaluated, we could say that, based on the learning curves, the best spatial density counting model is the *Siamese GMN*. However, since the loss values were quite similar, as shown in Figure 33 showing all spatial density learning curves, all models could achieve the same loss values or even better on a different training iteration or with a different validation set, therefore we can't conclude yet which model is the best. Because of that, the models were evaluated on a separate test set where the metrics mentioned in Section 5.2.3 have been used

to build the performance results on Table 4. Moreover, a visual comparison of their density maps can be observed in Figure 34 with multiple objects in the same image.

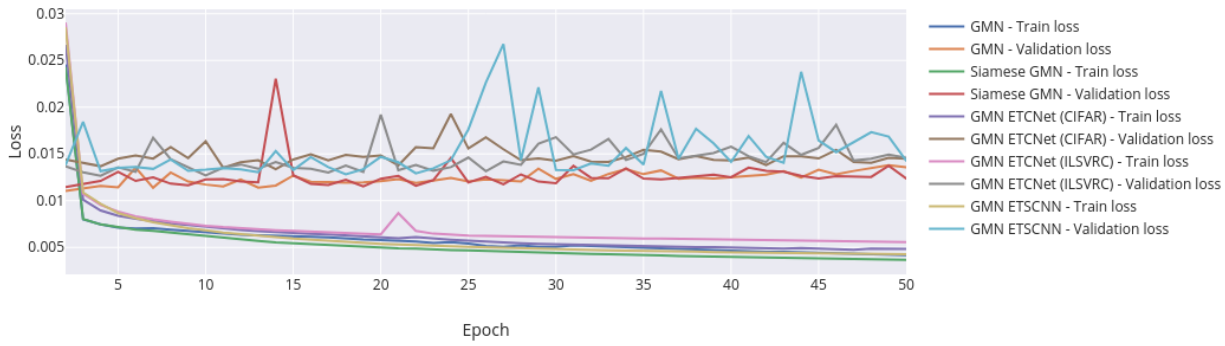


Figure 33: Performance comparison of all spatial density counting models through their learning curves.

Figure 34 presents the prediction of a sample image from the ILSVRC2015 dataset from each spatial density counting model, where the white stains represent the areas where an object has been located, and the red crosses are the local maximums extracted from this density map. From these predictions, we can clearly differentiate the models that are able to detect the elements of the image from the ones that don't. Starting from the least performant models, we can see how both *GMN ETCNet* and the *GMN ETSCNN* models which make use of a VAE trained on the ILSVRC dataset, were not able to detect the shape of the horses in the original image. Instead, it seems like they learnt the color of the horses since they can detect the darker parts of the image, such as its margins. This is probably due to the lack of definition from the encoded template, as shown in its decoding in Figure 29c. In contrast, the *GMN ETCNet* with the VAE trained on the CIFAR10 dataset, achieves better mapping of the given horse template but still matches the lower margin of the image. This fact proves that having better decoding of the Variational Autoencoder is directly related to the performance of the counting model due to having a more descriptive latent distribution. Thus, if we are able to define an unsupervised autoencoder that can provide a correct description of the given image, we will be able to achieve an encoding for the template that is unbiased from the training dataset and is able to achieve a better generalization on unseen environments since its features can't be overfitted. Different ways of obtaining better descriptive autoencoder will be mentioned in the future work mentioned in Section 7.

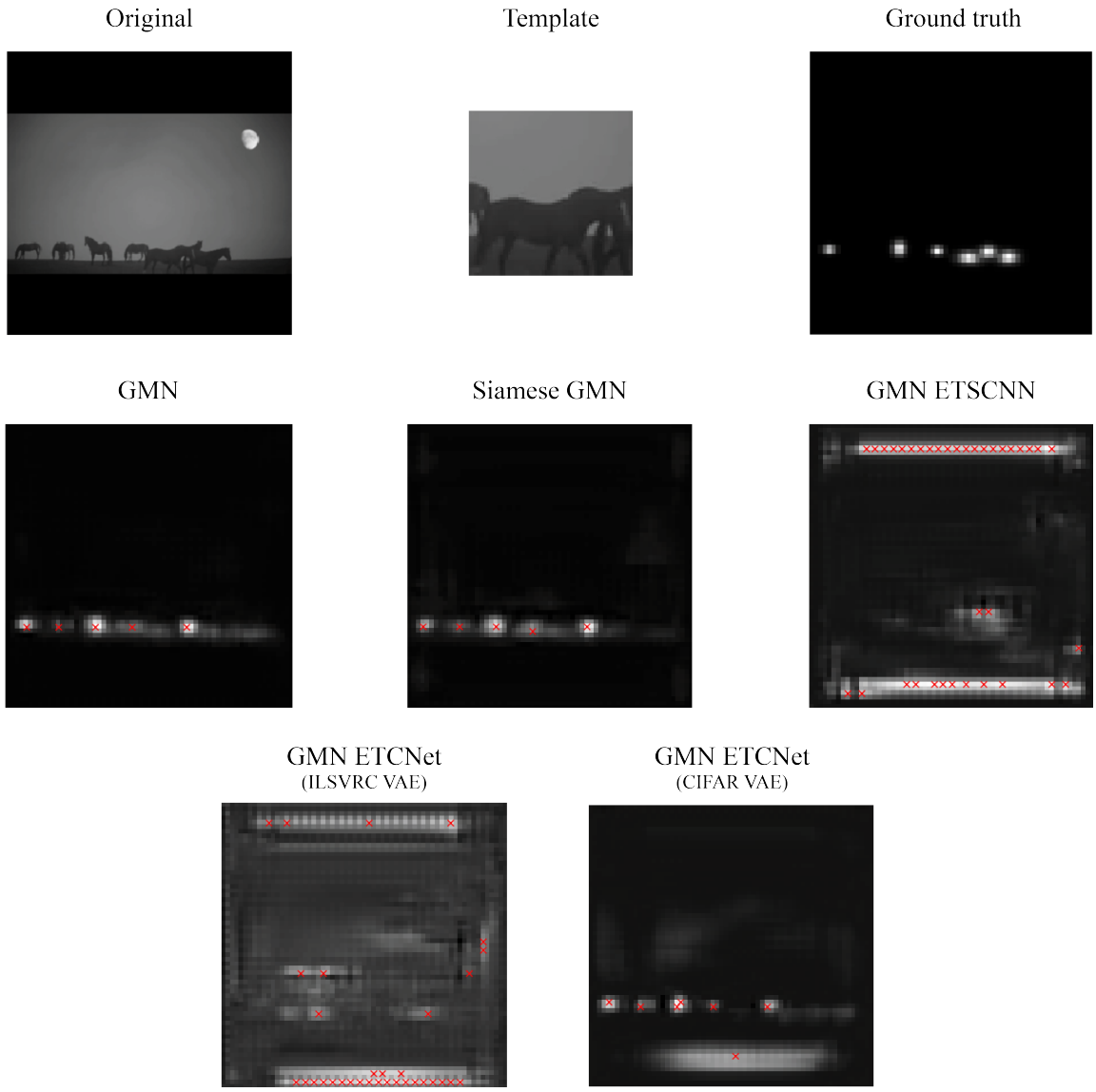


Figure 34: Predictions of the spatial density counting models on the ILSVRC2015 dataset.

In regards to the *Siamese GMN*, we can see how it is able to perform as good as the *Generic Matching Network* [15] with significantly less number of learnable parameters, as previously shown in Table 1. Comparing the result of only one predicted sample is not enough to draw conclusions, however, based just on this example, the model seems to be an improved version of the current state of the art in class-agnostic counting models, this hypothesis will be validated after evaluating their performance metrics on the test set. Besides, an interesting fact about the example shown in Figure 34 is that both the GMN and the *Siamese GMN* were able to detect the second horse instance starting from the left that was not correctly labeled in the original image proving, even more, their efficiency.

	MSE Loss	MAE Loss	Accuracy
Generic Matching Network [15]	274.1499	3.9452	16.8485%
Siamese GMN	43.0953	3.3857	12.1572%
GMN ETCNet (ILSVRC VAE)	384.9722	9.9310	3.1528%
GMN ETCNet (CIFAR VAE)	92.7227	6.2081	4.7570%
GMN ETSCNN	56.1181	4.6096	7.2978%

Table 4: Performance result of the spatial density counting models on the test set.

As previously mentioned, in order to extract conclusions for the comparison between models,

we can't rely on just one sample image. For this reason, Table 4 shows the performance metrics of each spatial density model on the test set. These metrics have been computed from the predicted count out of the local maximums in the image, therefore, the accuracy has been computed from those instances which were predicted correctly. In the case of those whose count was not predicted correctly the MAE represents the deviation of the predicted counts, *e.g.:* in the case of the Generic Matching Network which has a MAE Loss of 3.9452, it means that its prediction can be almost 4 units above or below the real value.

If we compare the results from each model in the test set, we can observe that the model that achieved the worst scores is the *GMN ETCNet* with the ILSVRC VAE. This was an expected outcome after observing the terrible prediction of the network in the previous example. After observing how bad these scores are, we can prove that this model is underfitted and is not able to achieve the right mapping of the template features in the original image.

With respect to its alternative configuration trained using the CIFAR10 Variational Autoencoder, we can see how its losses have been decreased with respect to the previous model. Since the trained VAE is the only difference between them, we can conclude both that the CIFAR10 VAE is more descriptive with respect to its template, as well as the fact that a more descriptive VAE improves the performance of the counting model.

The case whose results in the test set were surprising after the prediction shown in Figure 34 was that of the *GMN ETSCNN* model which, thanks to its siamese architecture, is able to achieve a better mapping than the previous cases. Nonetheless, after analyzing its predictions, we have observed that the model was able to achieve a correct prediction when the original image only contained a single instance of the given template, in all other cases where more than one instance was present, the model was not able to localize the features of the others, as in the previous example. This good prediction in the scenarios with one single instance compensates the error from the others, which explains the decrease shown in the losses in Table 4. Therefore, since the only difference between the *GMN ETCNet* and the *GMN ETSCNN* is its siamese architecture, the only explanation for this improvement in its performance is that using a siamese architecture not only decreases the number of learnable parameters but also helps in the feature extraction process from the *embedding* module. This fact is also validated after analyzing the loss decrease by the *Siamese GMN* with respect to the baseline model.

In the case of the *Siamese GMN*, it presented better values for the MSE Loss and the MAE Loss but a worse accuracy than the baseline model. Despite this, by obtaining a lower Mean Absolute Error, we can deduce that the predictions were more similar to the ground truth, *i.e.* better. Having said that, thanks to achieving better performance metrics and having a lower number of learnable parameters, we can conclude that **the best spatial density model evaluated in this project for class-agnostic counting is the *Siamese GMN*.**

Up to this point, we have analyzed the performance of the models in the environment they were trained in, which, even though it is quite broad, still lacks examples from multiple real scenarios. Because of that, the generalization skills of the models will be evaluated under the CARPK dataset in the following section.

6.2.1 Analysis on unseen data

This section will summarize the performance results of the spatial density counting models on an unseen environment to analyze their generalization skills, which is a key factor when talking about class-agnostic counting models.

The models have been evaluated with the same metrics as in the previous experiments on the test split for the CARPK dataset. These performance results will be compared against the current state of the art in class-agnostic counting that is the *Generic Matching Network*. Since the CARPK dataset has been modified to fit the spatial density counting task, any other model in the state of

the art used for object detection with the CARPK dataset could not be compared. Nonetheless, you can refer to Table 3 in the *Class-Agnostic Counting* paper [15] where the GMN has been adapted to compare it with other object detection models.

By observing the predicted density maps shown in Figure 35, we can see how most of the models (except the *GMN ETCNet with ILSVRC VAE*) are able to recognize where the cars are located but not with enough detail as in the ground truth. The same behavior was observed with all images from this dataset. This fact proves that the models can't generalize in unseen environments unless they are explicitly trained. However, just as the *Class-Agnostic Counting* [15] paper mentioned, these models can be used as an initialization on new environments and achieve promising results after being trained with a few samples.

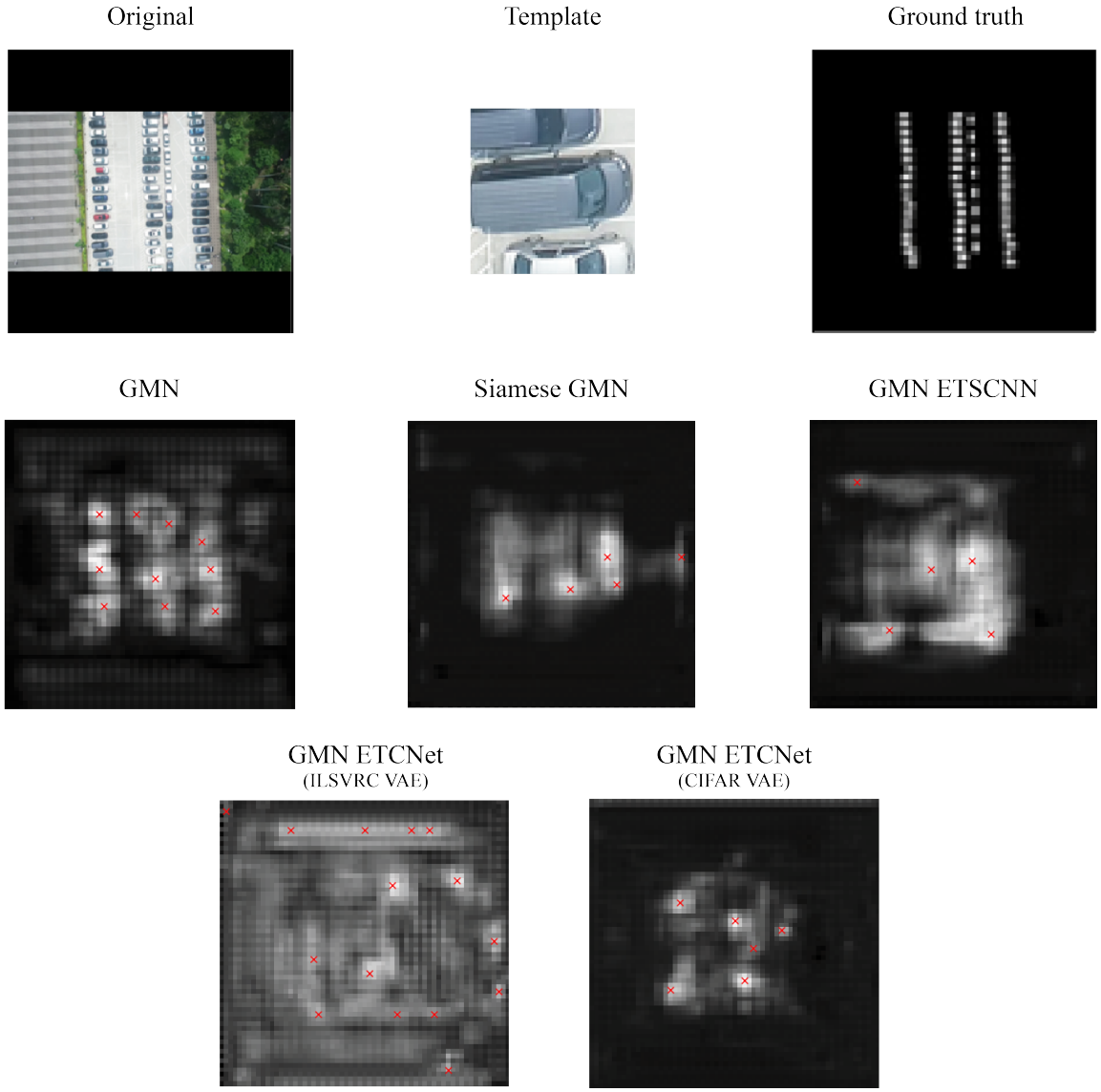


Figure 35: Prediction of the spatial density counting models on the CARPK dataset.

6.3 Counting strategy comparison

The goal of this project is to analyze the different approaches for object counting along with the proposal of new models for class-agnostic counting. This section provides a description of the advantages and disadvantages of each evaluated counting methodology based on the observed results in previous sections. In order to be able to compare both strategies, the same model design ideas have been applied in each approach.

Starting with direct counting models, the main advantage of using this approach is that they provide a single scalar value for the given image which prevents it from using any post-processing algorithm for computing the count from the local maximum values like its alternative. Furthermore, models following a direct counting strategy tend to be simpler due to the simplicity of the desired output. A clear example is the architecture of the model from *Deep People Counting in Extremely Dense Crowds* [25] shown in Figure 11 where a vanilla convolutional neural network has been used in a counting task. Nonetheless, these models force the model designer to have faith in the predicted count since the hidden layers of the network are what is known as a "black box" because the analyzed features are unknown. It is possible to analyze the evolution of the given input through the different layers of the network as well as the weight values of the convolution filters but, since we are working with multiple classes of objects in the same model, it is possible that the analyzed features are not interpretable from a human point of view. An alternative could be providing a scalar value for smaller regions of the image, like Stahl T. *et al.* did in [14], but we couldn't still be sure of what is it counting.

Moreover, based on the results observed in Section 6.1, we can conclude that learning a feature mapping in a class-agnostic direct counting approach is a task that is too abstract to be learnt directly from the model without any guidance by the model's architecture or behavior. None of the evaluated models were able to achieve a decent mapping, in fact, all predicted values from the models varied from 0.8 to 1, showing a severe underfitting even with a complex network architecture as it is the `ResNet-50`.

On the other hand, spatial density models offer an output that proves recognition of the objects in the scene. The output heat map, not only represents the counting of the elements but also their location, size and how clear was the recognition based on how high are the values on the predicted output. Additionally, when used on sequential instances, the density maps can also be used to analyze movement patterns from the objects in the scene. The only disadvantage is requiring an additional tool for converting the heat map into one scalar value, however, multiple techniques can be used for obtaining it based, for example, in counting the local maximums in the map or the values above a threshold. Therefore, counting by providing a spatial density distribution offers awareness of how the model is working which allows giving more confidence to its results.

The analysis presented in Section 6.2 demonstrated how networks with fewer parameters can achieve better results when the architecture of the model splits the responsibilities through different modules and guides the network into achieving a better mapping of its features. As observed, the trained models sometimes still had difficulties for predicting the density of the image to count from but, thanks to its output structure, it allowed validating that the predicted output was reasonable when compared against the ground truth instead of predicting a random number.

When comparing the results from both approaches we can observe how direct counting models have a fast convergence point, although we believe it is due to not being able to learn a mapping between the given template and the original input. Instead, abstracting the model from the counting task as the spatial density approach does, helps the model to focus on a pattern-matching task and generating a higher value when more patterns are matched.

From a model designing point of view, designing a direct counting model is a hard task due to not knowing which kind of information or tools is the model lacking since ... For this reason, and due to the complexity of the counting task, we believe it is a good practice to divide the counting goal into several steps and sub-goals to be completed before obtaining the final count. This process not only will help the model in learning the mapping between the image and the final count but also will allow adding a loss metric for each step improving the model performance (if the necessary supervised information is available).

As a conclusion, we can say that counting by regressing a density map seems a better approach based on both performance results and applications in a class-agnostic counting context.

7 | Discussion and Future work

This project aimed to analyze several object counting strategies seeking the most efficient approach to achieve a class-agnostic counting model. After studying the state of the art in automatic object counting, we observed how the best approach was by following a regression-based strategy since it presented the higher flexibility on counting in crowded areas. Based on that fact, we decided to analyze the two main ways of performing regression-based object counting (direct counting and spatial density counting) by comparing the performance of several deep learning models on each approach also with the idea of achieving a true class-agnostic counting model which doesn't have to be trained on unseen environments.

These models have been developed trying to mimic human behavior in object recognition, as well as using some common architectures that work well in other computer vision tasks. Each idea that defined the architecture of the models, has been used for developing a model in each counting strategy in order to keep a fair comparison between them.

After training each model, we were able to observe how the models following a direct counting methodology had more difficulties to learn the necessary feature mapping to count the objects in the image, even with a simple dataset such as the *Count CIFAR10*. In contrast, spatial density models achieved decent performance results after being trained with 30 different categories of objects. Moreover, the spatial density models were evaluated in an unseen environment for counting vehicles in parking lots to evaluate their generalization capabilities. Here, although they were able to detect the dense area where the cars were located, none of them achieved enough resolution to count how many cars are in the image without explicit training.

We believe that the improvement in the model's performance presented in the spatial density counting models is due to abstracting the model from the counting task allowing it to focus on the task of comparing both the given template and the image to count from, as well as locating each of the objects in the image. In fact, we think that the more split the functionalities of each component of the model are the better will be the final results, even more, if we are able to add a supervised loss value to each module and group them into one single loss value for the gradient computation.

Besides their better performance, the output structure of density maps are also more versatile and can be used in multiple applications such as movement pattern analysis, detection of abnormal situations by detecting small regions with a high density of objects, etc. Instead, providing only the total number of objects in the image provides only one type of information without a description of their distribution.

Having said that, we believe that the best approach to achieve a true class-agnostic counting model is by developing a regression-based spatial density counting model and by splitting the functionalities into separate modules. More specifically, this project has achieved to develop a deep learning model, the *Siamese GMN*, that improves the current state of the art in class-agnostic counting thanks to having a reduced number of learnable parameters and improving its prediction error. The idea behind this model is to use a single feature extraction network for both the template and the image to count from. This strategy allows having the same encoding scale for both images, which helps its later matching.

Even if not all developed models achieved a better performance of the current state of the art, this project has helped in investigating new architectures to approach the counting task and allowed to discard others that didn't work so well. One example would be the *GMN ETCNet* which didn't obtain a better loss but still evaluates an interesting idea of trying to achieve a feature extraction of the images that is unbiased from the counting dataset.

Moreover, another contribution of this project is the creation of different datasets that have been specially defined for training class-agnostic counting models. These datasets will be published for

being used by any other regression-based counting projects and to foment further research with the same goal as this one.

The results of this thesis have opened several next steps in the direction of achieving a true class-agnostic counting model. Among all the possible future work to be done, we wanted to highlight the following next steps that we think are worth investigating:

- Add the prediction of a regressed scalar value for the count apart from the density map combining both regression-based approaches.
- Adding a scalar input for describing the desired sensitivity for detecting the template. This way, the users would be able to specify how precise should be the matching on the original image, for example, given the template of a red car they could decide if they want to find cars in general or only red cars.
- Split the template into different regions and encode each one individually to improve its description.
- Change the output density map on the defined datasets to be heat maps where the final count would be the sum of the values on all pixels.
- Improving the description of the Variational Autoencoders by adding Discriminative networks to its training.
- Use the Variational Autoencoders directly as feature extractors.
- Developing more deep learning models that make use of other unsupervised neural networks for extracting the features from both images to abstract the counting model from the dataset which will provide an unbiased feature extraction that can be used in any environment.

There is so much work to be done in the direction of class-agnostic counting and we would like to encourage any other research project working in this direction.

References

[1] Change Loy, C., Chen, K., Gong, S., &; Xiang, T. (2013, October). Crowd Counting and Profiling: Methodology and Evaluation. Retrieved from: https://www.researchgate.net/publication/257927659_Crowd_Counting_and_Profiling_Methodology_and_Evaluation

[2] Maddalena, L., Petrosino, A., &; Russo, F. (2014, January 01). People counting by learning their appearance in a multi-view camera environment. Retrieved from: https://dl.acm.org/doi/10.1016/j.patrec.2013.10.006

[3] Sindagi, V., &; Patel, V. (2017, July 05). A Survey of Recent Advances in CNN-based Single Image Crowd Counting and Density Estimation. Retrieved from: https://arxiv.org/abs/1707.01202

[4] Marsden, M., McGuinness, K., Little, S., Keogh, C., &; O'Connor, N. (2017, November 15). People, Penguins and Petri Dishes: Adapting Object Counting Models To New Visual Domains And Object Types Without Forgetting. Retrieved from: https://arxiv.org/abs/1711.05586

[5] Xie, W., Noble, J. A., &; Zisserman, A. (2015, November 15). Microscopy cell counting and detection with fully convolutional regression networks. Retrieved from: https://www.tandfonline.com/doi/abs/10.1080/21681163.2016.1149104

[6] Xue, Y., Ray, N., Hugh, J., &; Bigras, G. (2016, October 08). Cell Counting by Regression Using Convolutional Neural Network. Retrieved from: https://link.springer.com/chapter/10.1007/978-3-319-46604-0_20

[7] Guerrero-Gómez-Olmedo, R., Torre-Jimenez, B., López-Sastre, R., &; Maldonado-Bascón, S. (2015, June). Extremely Overlapping Vehicle Counting. Retrieved from: https://www.researchgate.net/publication/281408569_Extremely_Overlapping_Vehicle_Counting

[8] Arteta, C., Lempitsky, V., &; Zisserman, A. (2016, October). Counting in the Wild. Retrieved from: https://www.researchgate.net/publication/308190185_Counting_in_the_Wild

[9] Fiaschi, L., Nair, R., Köthe, U., &; Hamprecht, F. (2012, January). Learning to count with regression forest and structured labels. Retrieved from: https://www.researchgate.net/publication/261130953_Learning_to_count_with_regression_forest_and_structured_labels

[10] Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., &; Shah, R. (1993, November 01). Signature verification using a "Siamese" time delay neural network. Retrieved from: https://dl.acm.org/doi/10.5555/2987189.2987282

[11] Oñoro-Rubio, D., Niepert, M., &; López-Sastre, R. (2018, November 15). Learning Short-Cut Connections for Object Counting. Retrieved from: https://arxiv.org/abs/1805.02919

[12] He, K., Zhang, X., Ren, S., &; Sun, J. (2015, December 10). Deep Residual Learning for Image Recognition. Retrieved from: https://arxiv.org/abs/1512.03385

[13] Lempitsky, V., &; Zisserman, A. (2010, January). Learning To Count Objects in Images. Retrieved from:

https://papers.nips.cc/paper/2010/file/fe73f687e5bc5280214e0486b273a5f9-Paper.pdf

[14] Stahl, T., Pintea, S. L., &; Van Gemert, J. C. (2018, October 10). Divide and Count: Generic Object Counting by Image Divisions. Retrieved from:
https://ieeexplore.ieee.org/document/8488575

[15] Lu, E., Xie, W., &; Zisserman, A. (2018, November 01). Class-Agnostic Counting. Retrieved from:
https://arxiv.org/abs/1811.00472

[16] Mehmood, A., Maqsood, M., Bashir, M., &; Shuyuan, Y. (2020, February 5). A Deep Siamese Convolution Neural Network for Multi-Class Classification of Alzheimer Disease. Retrieved from:
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7071616/

[17] Bertinetto, L., Valmadre, J., Henriques, J., Vedaldi, A., &; Torr, P. (2016, September 14). Fully-Convolutional Siamese Networks for Object Tracking. Retrieved from:
https://arxiv.org/abs/1606.09549

[18] Dey, S., Dutta, A., Toledo, J., Ghosh, S., Llados, J., &; Pal, U. (2017, September 30). SigNet: Convolutional Siamese Network for Writer Independent Offline Signature Verification. Retrieved from:
https://arxiv.org/abs/1707.02131

[19] Hsieh, M., Lin, Y., &p; Hsu, W. (2017, August 03). Drone-based Object Counting by Spatially Regularized Regional Proposal Network. Retrieved from:
https://arxiv.org/abs/1707.05972

[20] Saqib, M., Khan, S. D., Sharma, N., &; Blumenstein, M. (2019, March). Crowd Counting in Low-Resolution Crowded Scenes Using Region-Based Deep Convolutional Neural Networks. Retrieved from:
https://ieeexplore.ieee.org/document/8669755

[21] Liu, J., Gao, C., Meng, D., &; Hauptmann, A. (2018, March 07). DecideNet: Counting Varying Density Crowds Through Attention Guided Detection and Density Estimation. Retrieved from:
https://arxiv.org/abs/1712.06679

[22] Zhang, Y., Zhou, D., Chen, S., Gao, S., &; Ma, Y. (2016, June). Single-Image Crowd Counting via Multi-Column Convolutional Neural Network. Retrieved from:
https://ieeexplore.ieee.org/document/7780439

[23] Boominathan, L., Kruthiventi, S., &; Babu, R. (2016, August 22). CrowdNet: A Deep Convolutional Network for Dense Crowd Counting. Retrieved from:
https://arxiv.org/abs/1608.06197

[24] Oñoro-Rubio, D., &; López-Sastre, R. (2016, October 08). Towards Perspective-Free Object Counting with Deep Learning. Retrieved from:
https://link.springer.com/chapter/10.1007/978-3-319-46478-7_38

[25] Wang, C., Zhang, H., Yang, L., Liu, S., &; Cao, X. (2015, October 01). Deep People Counting in Extremely Dense Crowds. Retrieved from:
https://dl.acm.org/doi/10.1145/2733373.2806337

[26] Kingma, D., &; Ba, J. (2017, January 30). Adam: A Method for Stochastic Optimization. Retrieved from:
https://arxiv.org/abs/1412.6980

[27] Rebuffi, S., Bilen, H., &; Vedaldi, A. (2018, March 27). Efficient parametrization of multi-domain deep neural networks. Retrieved from:
https://arxiv.org/abs/1803.10082